C3.6/2:C33/4

CENFOR

CENSUS FORTRAN MANUAL

DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS
MARCH 1971

PENNSYLVANIA STATE UNIVERSITY
APR 29 1971
DOCUMENTS COLLECTION

PREFACE

Elements of the FORTRAN language and the implementations for data processing at the Bureau of the Census on UNIVAC equipment are presented in this manual.

Census subroutines and functions, reference lists, examples, exercises and work procedures are included.

Additions and corrections to this manual will be made as required or recommended.

UNIVAC source manuals are listed in the bibliography for this manual.

Bureau of the Census
1971

U. S. Depository Copy

Digitized by the Internet Archive
in 2012 with funding from
LYRASIS Members and Sloan Foundation

http://archive.org/details/cenforcensusfort00unit

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Ever Wonder How A Computer Works?
1.1.1. Hardware vs. Software
1.1.2. Input Devices
1.1.3. Computer Language
1.1.4. Units of Information
1.1.5. Output Devices

1.2. Introduction to FORTRAN Programing

1.3. Internal Representation
1.3.1. Words

1.4. Number Systems
1.4.1. Binary
1.4.2. Octal
1.4.3. Number Conversion
1.4.4. Arithmetic in other Bases
1.4.5. Conversion
1.4.5.1. Binary to Octal
1.4.5.2. Octal to Binary
1.4.5.3. Decimal-Octal Conversion
1.4.5.4. Binary-Decimal Conversion
1.4.5.5. Converting Fractions
1.4.5.6. Mixed Numbers
1.4.6. Signed Binary Numbers
1.4.7. Floating Point
1.4.8. Binary Coded Decimal Codes
1.4.9. Logical Operations

1.5. The 1107 Central Computer
1.5.1. The Control Portion
1.5.2. The Computer Memory
1.5.3. The Arithmetic Section
1.5.4. Input-Output Channels
1.5.5. Peripheral Equipment

1.6. Exercises

TABLE OF CONTENTS (Continued)

2. THE FORTRAN LANGUAGE FUNDAMENTALS

2.1. Writing FORTRAN Programs
2.1.1. Basic Characters
2.1.2. Preparation of Programs and Related Materials for Key Punching
2.1.3. Source Statements

2.2. Constants
2.2.1. Integer-Fixed Point
2.2.2. Real-Single Precision Floating Point
2.2.3. Double Precision-Double Precision Floating Point
2.2.4. Complex
2.2.5. Logical

2.3. Variables

2.4. Operators

2.5. Expressions
2.5.1. Integer Division
2.5.2. Limitations of Arithmetic Operations
2.5.3. Function Reference

2.6. Arithmetic Assigment Statements

2.7. Arithmetic IF

2.8. Logical Expressions

2.9. Logical Statements

2.10. Logical IF

2.11. PARAMETER Variables

2.12. CONTINUE Statement

2.13. GO TO
2.13.1. The Unconditional GO TO
2.13.2. ASSIGN Statement
2.13.3. Assigned GO TO

2.14. Mathematical Functions

2.15. STOP

2.16. END

2.17. Exercises

TABLE OF CONTENTS (Continued)

3. SUBSCRIPTED VARIABLES

3.1. Subscripts
3.1.1. Allowable Forms of Subscripts
3.1.2. Rules for Subscript Formation
3.1.3. Valid Subscripts Used with Array Elements
3.1.4. Invalid Subscripts Used with Array Elements

3.2. Arrays
3.2.1. One-Dimensional Array = "LIST"
3.2.2. Two-Dimensional Array = "TABLE"
3.2.3. Three-Dimensional Array = SET OF TABLES

3.3. Tables and Files

3.4. Dimension Limits

3.5. Mathematical Notation

4. THE DO STATEMENT

4.1. DO

4.2. DO-LOOPS

4.3. Nested DO-LOOPS

4.4. Materialization of the Index Variable

4.5. Frequent Uses of a DO-LOOP

4.6. Exercises Using the DO Instruction

5. FORTRAN I/O

5.1. Formatted READ and WRITE
5.1.1. Formatted READ
5.1.2. Note on Method to Detect End-Of-Data on Tape
5.1.3. Formatted WRITE
5.1.4. Control of Print Files
5.1.5. READ and WRITE Name Lists

5.2. Implied DO

5.3. UNIT Assignments for FORTRAN READ/WRITE Statements

5.4. FORMAT Statement
5.4.1. Field Descriptors
5.4.2. Table of Editing Symbols

TABLE OF CONTENTS (Continued)

5.4.3. Object Time FORMAT Statements
5.4.4. Rescanning
5.4.5. Specific Information Re Editing Codes

5.5. Tape Positioning Statements
5.5.1. END FILE
5.5.2. REWIND
5.5.3. BACKSPACE

5.6. UNFORMATTED Input and Output

5.7. Printing a Formatted Tape

5.8. Exercises

6. FUNCTION AND SUBROUTINE PROCEDURES

6.1. General

6.2. Procedure Arguments

6.3. The FUNCTION
6.3.1. Referencing a Function
6.3.2. Statement Functions
6.3.3. Examples of Statement Functions
6.3.4. Built-In Functions
6.3.5. Note on MINO, MIN1, AMINO, MAXO, MAX1, AMAXO, AMAX2
6.3.6. Note on MOD, AMOD Function
6.3.7. Built-In Typeless Functions AND, OR, LOC, COMPL, BOOL
6.3.8. External and Internal Functions
6.3.9. Note on SIN, DSIN, CSIN
6.3.10. Subprograms
6.3.10.1. External Subprogram-General
6.3.10.2. Function Subprograms-External
6.3.10.3. RETURN
6.3.10.4. Examples of Function Subprograms

6.4. Subroutine Subprogram-External

6.5. ABNORMAL Functions

6.6. EXTERNAL Statement

6.7. Errors to be Avoided When Using Subprograms

6.8. Exercises

TABLE OF CONTENTS (Continued)

7. SPECIFICATION STATEMENTS

7.1. General

7.2. DIMENSION Statement
7.2.1. Adjustable Dimensions

7.3. EQUIVALENCE Statement

7.4. COMMON Statement
7.4.1. General
7.4.2. COMMON Block Size Rule
7.4.3. COMMON Used with EQUIVALENCE
7.4.4. Some Uses of COMMON

7.5. DATA Statement

7.6. Exercises

8. CENSUS LIBRARY FUNCTIONS

8.1. LPM_m (x)

8.2. J_{jf} (NAME,n)

8.3. $IXTRC_j$ (NAME,n)

8.4. LS_{tt} (NAME,n)

8.5. LSSU (NA,MASK,NB)

8.6. LADD (NA,NB)

8.7. LSUB (NA,NB)

8.8. LAND (NA,NB)

8.9. LFX (NAME)

8.10. LXF (NAME)

8.11. LFB (NAME)

8.12. LXB (NAME)

8.13. LBF (NBIN,NFILL)

8.14. LBX (NBIN,NFILL)

8.15. MSR (4Hxxxx)

TABLE OF CONTENTS (Continued)

9. CENSUS LIBRARY SUBROUTINES

9.1. CHKDIV ($n)

9.2. NOCHK

9.3. TYPE (n,nHMESSAGE)

9.4. TYPE (n,nHMESSAGE,j,JREPLY)

9.5. LOAD (u,n,nHMESSAGE, $999)

9.6. CHANGE (u,n,nHMESSAGE, $999)

9.7. INSRTf (NA,NB,n)

9.8. INSERT (NA,NARAY,n)

9.9. LDStt (NA,NB,n,NC,ND)

9.10. MOVEf (NA,n,NB,n)

9.11. LB2X (NBIN,NXS3,NFILL)

9.12. LB2F (NBIN,NFD,NFILL)

9.13. UNPK (INPUT,EDIT,CORE)

9.14. PACK (CORE,EDIT,OUTPUT)

9.15. SORT (LFILE,LPAR) - An Internal Sort

9.16. EX: A FORTRAN Input/Output Subroutine

9.17. Exercises

10. FOLION

10.1. LION Formats

10.2. FOLION
10.2.1. Introduction
10.2.1.1. File Labels
10.2.1.1.1. IIIA Labels
10.2.1.1.2. IIA Labels
10.2.2. Output Subroutine
10.2.2.1. Open Output File
10.2.2.2. Write Output File
10.2.2.3. Close Output Reel
10.2.2.4. Close Output File

TABLE OF CONTENTS (Continued)

10.2.3. Input Subroutine
10.2.3.1. Open Input File
10.2.3.2. Read Input File
10.2.3.3. Close Input Reel
10.2.3.4. Close Input File
10.2.3.5. Note on IIIA End-of-Reel Exit
10.2.4. File Control Errors
10.2.5. A Coded Example
10.2.6. FORTRAN I/O Equipment Configuration

10.3. Exercises

11. GENERAL INFORMATION

11.1. Planning a HSP Output Table

11.2. Data Record Layouts

11.3. Creating Test Decks

11.4. Tape Blanking and Storage

11.5. Schedule Cards
11.5.1. Priority Card (PTY)
11.5.2. Facility Description Card (FAC)
11.5.3. Table Length Assignment Card (TAL)
11.5.4. Transfer Card (TRN)
11.5.5. Subroutine Card (SUB)
11.5.6. Parameter Cards (PMn)

11.6. Submission of Programs
General Instructions and Forms Required

11.7. Programer Responsibilities

11.8. Exercise

12. ASSEMBLY, CORRECTING, AND DEBUGGING

12.1. Assembly Control Cards
12.1.1. The START Card
12.1.2. The MAP Card
12.1.3. The FOR Card
12.1.4. The STOP Card
12.1.5. Examples of Control Cards

12.2. Program Corrections

12.3. FORTRAN Debugging

TABLE OF CONTENTS (Continued)

13. THE TAPE SORT

13.1. Using FORTRAN Generated Subprograms as Own Coding to the 1107 Sort-Merge
13.1.1. Concept of "OWN CODING".
13.1.2. Coding the OCFP and OCLP Subprograms
13.1.3. Entrances to Your Own Coding from the S/M
13.1.4. Returns from Your Own Coding to the S/M
13.1.5. I/O Statement Within the Own Coding Subroutines
13.1.6. Communication Between OCFP and OCLP Subprograms

13.2. Scheduling for SORT
13.2.1. PTY Card
13.2.2. FAC Card
13.2.3. TAL Card
13.2.4. SUB Card
13.2.5. PM0 Card
13.2.6. PM1 Card
13.2.7. PM 3,4,5 Cards
13.2.8. PM 6,7,8 Cards
13.2.9. Example of SORT Schedule Cards

13.3. Determination of CORE, DRUM and Tape Requirements for a SORT

13.4. Information Relative to the Straight Merge Only

13.5. Significant Console Messages

13.6. Notes on SORT/MERGE

14. SERVICE PROGRAMS

14.1. CENSER

14.2. CKSUM

14.3. SELTPR

14.4. AUTOCP

14.5. BTP

14.6. TCOPY

14.7. CTSIM

14.8. EDTCPY

14.9. CARTAP

14.10. DATACO

14.11. COPIER

TABLE OF CONTENTS (Continued)

14.12. RUTT

14.13. UTT

14.14. RUT702

14.15. UTT4C

14.16. MICRO

14.17. RUTA19

14.18. SALVAG

14.19. 2 to 3

14.20. 4C to 4C

14.21. TP4C

14.22. SUTT1

14.23. PSTRIP

14.24. FOIST

14.25. SR2FOL

14.26. MDMIC

14.27. RUTMDS

14.28. IMAGER-TWOPAS

14.29. EXERCISES

15. DECISION TABLES

15.1. Limited-Entry Decision Tables

15.2. Extended-Entry Decision Tables

15.3. Mixed-Entry Tables

15.4. Determining Maximum Table Size

15.5. A Method of Eliminating Irrelevant Tests

15.6. Bifurcated Table Arrangement

15.7. The ELSE Rule

TABLE OF CONTENTS (Continued)

15.8. Sequence Control Definitions

15.9. Examples of Decision Tables

15.10. Machine Processing of Decision Tables

15.11. TAB4IX: A Decision Table Preprocessor for the UNIVAC 1107 FORTRAN IV Compiler

16. PROGRAMING TECHNIQUES AND TIPS

16.1. FORTRAN: Externally Expandable Arrays Using Named COMMON

16.2. Compiling Large FORTRAN Programs

16.3. Defining Subroutine Formal Arguments

16.4. Multiply/Divide Sequence

16.5. Real to Integer Conversion

16.6. Integer to Double Precision Function

16.7. Table Look-up Techniques

16.8. A Subroutine for Sorting Words in an Array

16.9. A Random Number Generator Routine

16.10. A Subroutine to Create a Check Digit

16.11. Method Used to Create Check Digit

16.12. A Subroutine to Convert Milliseconds to Hours, Minutes, and Seconds in XS3

16.13. ZERO Subroutine

16.14. Permissible Calls Using Programer's Subroutine ZERO

16.15. Sequence Checking

16.16. SEARCH: Programer's Subroutine for Binary Search

16.17. DRUM USE

17. PROGRAM EXAMPLES

17.1. General

17.2. "PAYROLL" Specifications

TABLE OF CONTENTS (Continued)

17.3. "PROBABILITY of Purchasing an Automobile" Specifications

17.4. SORT with Option Coding

17.5. Specifications for Marriage Table 7A

18. FORTRAN V

18.1. INCLUDE

18.2. DELETE

18.3. EDIT

18.4. INPUT/OUTPUT END CLAUSE

18.5. INPUT/OUTPUT ERR CLAUSE

18.6. INPUT/OUTPUT STATUS WORD INSTAT FUNCTION

18.7. ERROR WALK BACK

18.8. EXTERNAL SUBPROGRAM

18.9. MULTIPLE ENTRY POINTS

18.10. AUTOMATIC TYPING

18.11. FLD FUNCTION

18.12. Define

18.13. Mixed Type Arithmetic

18.14. 1108 Double Precision Hardware

18.15. Parameter Variables

18.16. Data in Type and Dimension Statement

18.17. Implicit

18.18. Namelist Input/Output

18.19. Format Scan Pointer T

18.20. G Editing Code

18.21. Reread

18.22. Free Format Input/Output

TABLE OF CONTENTS (Continued)

18.23. Mass Storage Files

18.24. Encode/Decode

18.25. Optimization

18.26. Master Space Comment

18.27. Ru Format Code--Right-Justified Alphanumeric

18.28. Substitution for uH

18.29. Compiler Statement

18.30. Math-Pack and Stat-Pack

19. REFERENCE INFORMATION

19.1. Source Program Ordering

19.2. Limitations

19.3. Octal-Decimal Integer Conversion Table

19.4. 1107 Operator Termination Codes

19.5. Error Codes
19.5.1. LION Error Codes That Indicate Equipment Malfunction
19.5.2. LION Error Codes
19.5.3. Job Request Error Codes
19.5.4. Load Error Codes
19.5.5. EXEC I/O Error Messages
19.5.6. Termination Codes
19.5.7. Sort Error Codes
19.5.8. ELF Error Codes
19.5.9. FILE Control Errors

19.6. Channel Assignments
19.6.1. CENSUS 1107 Channel Assignments --1107A-B
19.6.2. CENSUS 1108 Channel Assignments --1108C
19.6.3. CENSUS 1108 Channel Assignments --1108D

19.7. UNIVAC 1107 Standard Code Translation

19.8. BCD, Excess-Three Codes

19.9. Table of Powers of 2

19.10. Capacity of Tapes Written With FOLION

19.11. Standards for Operation and Use of UNIVAC 1107 System

19.12. Most Used Boards and Loops for OFFLINE HSP

TABLE OF CONTENTS (Continued)

19.13. HSP Non-Printing Characters

19.14. Amount of Space per Page

19.15. Data Processing Equipment at the Bureau of the Census

19.16. Character Set for Information Interchange

19.17. Flowchart Symbols

19.18. Names to be Avoided for External Subroutines

19.19. LION Formats

19.20. FILE Control Table - Output

19.21. FILE Control Table - Input

TABLE OF CONTENTS (Continued)

20. SLEUTH II

20.1. Introduction to SLEUTH II

20.2. Instruction Word Format

20.3. The Label Field

20.4. The Operation Field

20.5. The Operand Field

20.6. Continuation

20.7. Termination

20.8. Data Word Generation

20.9. Expressions

20.10. Operators

20.11. SLEUTH II Assembler Directives
20.11.1. EQU
20.11.2. RES
20.11.3. FORM
20.11.4. END
20.11.5. LIT
20.11.6. INFO
20.11.7. DO
20.11.8. PROC
20.11.9. NAME
20.11.10. GO
20.11.11. FUNC
20.11.12. WRD
20.11.13. CHAR
20.11.14. LIST, UNLIST
20.11.15. EQUF

20.12. Communication Between FORTRAN and SLEUTH II
20.12.1. Through Use of COMMON Areas
20.12.2. Through Use of ARGUMENT Lists

20.13. APPENDIX C. INSTRUCTION REPERTOIRE

21. GLOSSARY

BIBLIOGRAPHY

1. INTRODUCTION

1.1. Ever Wonder How A Computer Works?

In the barely 25 years that computers have had an impact on our society, they have brought about a basic transformation in the way man lives. This ever increasing impact is evident in the areas of banking and business, science and medicine, education and entertainment. In education alone, there has been a change in the philosophy of teaching as well as a change in the content of what today's children are learning. Behold the parents staring with uncomfortable lack of familiarity at their third-grader's arithmetic homework.

This era of speeds and volumes beyond human understanding, of superlatives and extremes is created in part by the availability of computers. Without computers, however, there would be no possible way to keep up with processing, analyzing, and producing the information necessary to man's life. Indeed, the role of the steam engine in the industrial revolution finds its contemporary counterpart in the computer and the "information explosion." Just as machines were designed to reduce the workload of many people or animals, the need for automation of intellectual work came with the development of science and the increased amount of knowledge man must deal with every day. Thus the computer must be considered as a universal information-processing machine.

1.1.1. Hardware vs. Software

A computer consists of both a collection of electronic parts that is called hardware and a set of written procedures for controlling the hardware called software. In terms of hardware, a computer consists of input and output devices, arithmetic and control circuits, and a memory. The software is the program of instructions that puts the system to work and is written by a programer who analyzes the task at hand. Thus, any calculation that can be performed by any machine can be performed by a computer if there is a program directing it.

Confronted with what may appear to be the frenzied motion of input-output devices and the blinking of console lights, it is not hard to view the programer as the witch doctor of a black art. In fact, anyone who has ever written a recipe or a set of directions to perform a task or planned a route of travel has, to some extent, written a computer-like program.

1.1.2. Input Devices

The computer accepts information from its environment through its input devices; it combines this information, directed by the rules of the program stored in its memory, with other information also stored in its memory, and sends information back to its environment through its output devices. The information may be punched into cards or tape; it may be recorded magnetically

on tapes, keyboards, disks, drums, or cards; it may be speech in the form of electrical pulses transmitted across telephone or telegraph lines, or it may appear on a TV-like picture tube. In each case, there will be a device specifically designed to translate the information from the incoming mode to the language of the computer or from the machine language to a mode preferred by man.

The human brain performs a similar function by accepting information through its senses, combining it with information stored in the memory and translating it to an appropriate action, speech, etc., which returns to man's environment. However, computers perform operations a million to a billion times faster than humans and this increase in speed means a greater capacity to manipulate information on a large-scale basis.

1.1.3. Computer Language

If a computer is to be used, it must obviously be able to communicate with the outside world. For human-to-computer communication the colorful, expressive language of human-to-human communication is not necessary; it is not even desirable. Thus, although there are numerous "programing languages" (e.g. COBOL, FORTRAN), they are a simplification of the language one would use to describe a task to a friend. There is even a greater simplification in the machine language by which the computer operates internally.

1.1.4. Units of Information

The basic unit of information with which these machines work is the binary digit, so-called "bit." In binary notation, only two symbols are used, 1 and 0. This system of representation is particularly convenient for the recording and processing of information in electronic devices because they most easily operate in only two different ways. For instance, a wire is either conducting current or it is not; a switch may be open or it may be closed. Therefore, one of the binary symbols can be equated with the electronic condition "switch open," or "no current on the wire," while the other symbol can be equated with the opposite electronic condition "switch closed" or "current on the wire." By grouping bits in some agreed upon unit, meanings can be assigned to the various states that the unit (called a register here) may assume. For example, if a unit consisting of five elements is chosen, each element capable of being in one of two states (represented by 1 and 0) then there are a total of 32 states.

The register can assume any one of the combinations from 00000 to 11111. Specific and unique meanings can now be assigned to each of these states; perhaps the familiar decimal numbers 0 to 31 for example:

Binary Representation	Decimal Equivalent
00000	0
00001	1
00010	2
00011	3

and so forth. Perhaps the various combinations might be designated as the 26 letters of the alphabet with six combinations left over for punctuation symbols.

What these combinations of 1's and 0's will represent is agreed upon before the equipment is ever designed. Thus the language the computer uses internally--as well as the kinds of input-output modes to be used by the programer--is determined by the individuals designing and building the machine.

1.1.5. Output Devices

Information is most frequently prepared in such a way as to facilitate the translation between the outside environment and the computer. It may be produced by the computer itself through the output devices, or it may be by some auxiliary equipment such as a keypunch. Thus, a hole punched into a punched card can be interpreted as a 1, and the fact that there is no hole interpreted as a 0. A magnetic tape device will interpret a magnetically recorded spot on the tape as a 1, the absence of a magnetic spot as a 0.

Not only does one have the ability to represent information in binary notation, but it is also possible to perform arithmetic operations upon it, following a few simple rules. It turns out that the calculations one wants to do in a computer can be done by repeated binary additions, so addition is really the basic operation of computers.

Thus it is by converting information to this binary representation, storing it and operating on it with elements that can change state in a few millionths or billionths of a second, and finally converting it to a selected output mode, that the computer can process and produce information that affects all phases of man's life.

1.2 INTRODUCTION TO FORTRAN PROGRAMING

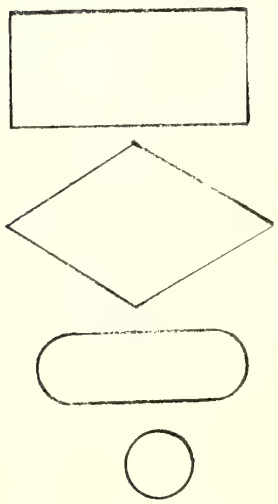
The process of writing a program should consist of a number of basic steps.

(1) First, the problem should be analyzed carefully to determine exactly what information is wanted. Then it must be determined which part of the known information must be used to resolve the problem. For instance, if the problem is to find the area of a triangle, one must know (or be able to find) the base and the height of that triangle.

(2) After the problem has been analyzed generally, the specific sequential steps must be determined. The most widely accepted method for planning the sequence of operations in a program is through flowcharting, a picture of the flow of the action of the program. In any problem that is even moderately complex, the interrelationships within it are difficult to keep clearly in mind without some visual representation. A flowchart provides this assistance.

A flowchart is made up of a set of boxes, the shapes of which indicate the nature of the operations described in the boxes, along with connecting lines that show the "flow of control" between various operations.

For our purposes here, the notation, which is quite simple, contains the following symbols:



A rectangle indicates any operation except a decision.

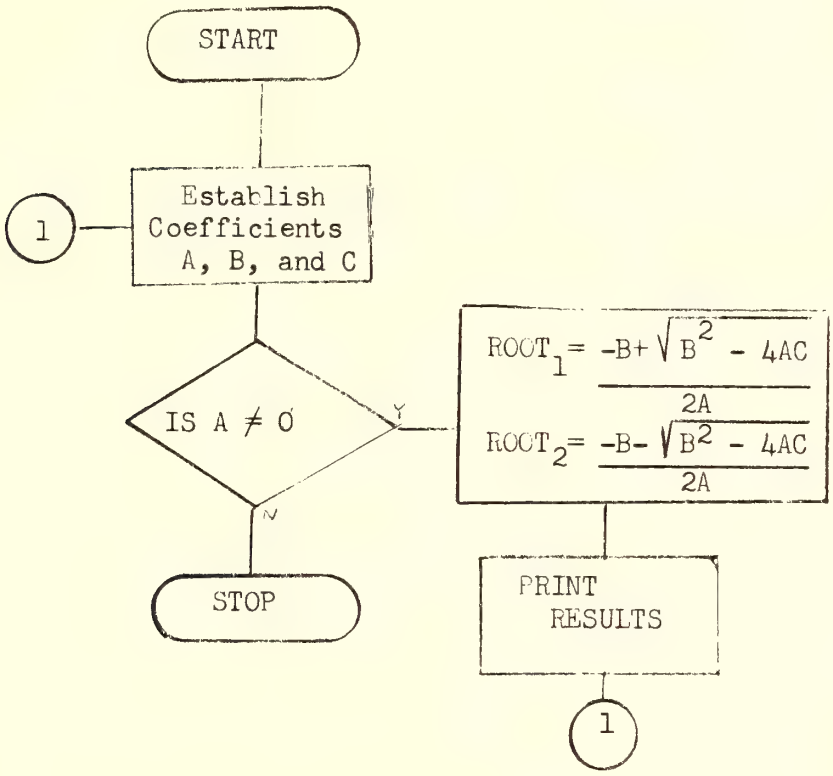
A diamond indicates a decision. If the decision is YES or TRUE, the action follows the line on the right. If the decision is NO or FAISE, the action follows the downward line.

An oval indicates the beginning, ending, or interruption point in a program.

A circle indicates the connection between two points in a flowchart.

The booklet "UNIVAC FLOWCHART SYMBOLS" describes the standards for flowchart symbols published in 1965 by the American Standards Association. Employment of the standard symbols is suggested to stay in tune with the computer industry. Booklets and templates are available in The Training Branch. Refer to 19.17. for a more complete description of the standard symbols.

The following is a flowchart for solving the quadratic equation. As in ordinary arithmetic, division by zero will produce an error, so any input which could cause such an error should be eliminated prior to its computation.



Flowcharts should always be written and read from left to right and from top to bottom.

Another method for schematically representing the flow of the control is the use of decision logic tables, tables which show the various paths which may be taken after a decision is made. Refer to chapter 15 "DECISION TABLES".

Decision Logic Table Example:

START	Y	N	N
$A \neq 0$		Y	N
ESTABLISH A,B,C	X		
Root 1 = (- B + $\sqrt{B^2 - 4AC}$) /2A		X	
Root 2 = (- B - $\sqrt{B^2 - 4AC}$) /2A		X	
PRINT		X	
RETURN	X		
STOP		X	X

While writing the flowchart or decision logic table, it must be remembered that a program must be written to consider every possible case. If a path is set if A<Ø and one is set if A>Ø, but if A = Ø, then the computer will not know what to do. No possible combination of parameter or eventualities can be omitted from the analysis, for the computer must be "told" everything! It is simply a mechanical and electronic device and "knows" nothing.

Whichever tool for planning the sequence of operations is used, the important point is that some considerable effort should be expended in planning the operation in order that it may enjoy a favorable prospect for success. It is most wastefull of programer effort and computer time for an operation to hang up for some error and require debugging. The use of the flow chart or the decision logic table helps insure that all possibilities are considered. In addition, they help point up similarities which can be handled in a generalized manner such as by the use of loops or subroutines. Generalized handling conserves effort and insures some freedom from error.

Another means of insuring the success of a program is through the use of modular Programing. A large program may be logically segmented into many small subprograms, each with its own duties. It is much easier and quicker to assemble and to debug these sections of logic than to assemble and debug one immense program. When all the parts have been completed, tha main program becomes merely a collection of references to its subprograms. Another feature of subprograms is that whenever a useful routine has been developed, it may be placed in the Library, and it may then be referenced by any program. If one program for sorting a list of words has already been written, it is much more economical to reference that program every time a sort is needed, rather than recoding the entire procedure.

Yet another means for effective programing is for programers to work in pairs or at least to communicate with one another in considerable detail. This implies the use of a common language and extensive documentation. The development of a program and its actual coding is most successfully achieved by the exchange of ideas and by help in checking logic and in the actual usage and coding of the programing language. The programing task is often much too difficult and complex to entrust to one person.

These three procedures (1) generalized programing, (2) modular programing, and (3) team programing are the basis for writing error-free programs.

(3) After an effective flowchart or decision logic table has been drawn, it is quite simple to follow its steps to create the sequence of statements, the coding, to be sent to the computer. But it must be remembered that the actual program statements themselves must be written with precision of execution, for a

single error in a single statement invalidates the entire program. These coded statements, printed on a FORTRAN coding sheet are then sent to be punched, one statement per card. You need not worry about the program again until the output returns to you, but meanwhile these punched cards are sent through a process which puts your coding on a tape. The computer, if fed the tape and its compiler, translates your coding into machine language, the language which the computer can interpret. It follows your sequence of instructions, using the data which must be supplied and returns the answers which you requested. At this time, hopefully, you have finished your program; but more realistically you must review what you have coded, make corrections, and resubmit your program.

(4) Even correct results printed by the computer do not always constitute a final answer to the problem. The user of the results must finally interpret the results to see what they mean in terms of the combinations of goals that the proposed program must satisfy. The customer often must use a trial and error approach to his problem.

(5) When the program works successfully, or during some other phase of writing the program, comments explaining how and why each step has occurred should be added. This documentation makes it easier both for you to follow your work at some later time, and for other people to understand what you have done. The major reasons for requiring documentation are: (1) in case the original programer is unable to complete the job, (2) so that future modifications may be added to the program, and (3) to make corrections if latent errors are uncovered during production processing. As much as is possible of the documentation should be done concurrently with the preparation of the program. This is especially true if the programer is working on different stages of one job, or on programs for more than one job.

Remember to do each step of programing thoroughly and carefully:

(1) PLANNING

(2) FLOWCHARTING OR WRITING DECISION LOGIC TABLES

(3) CODING

(4) DEBUGGING

(5) DOCUMENTATION

1.3. INTERNAL REPRESENTATION

The memory of a computer contains a large number of boxes, the storage locations, each having a distinct name, the address. Information may be put into each location, but only one word of information may be there at any time. When new information is placed in a location, the old information is destroyed. When the computer is told to multiply two numbers, it actually is told to multiply the contents of two addresses. It finds out what is in the first address, and copies this into the Arithmetic Unit. Then it multiplies what is in the second address by the first quantity, and then this result is placed into the specified address in memory.

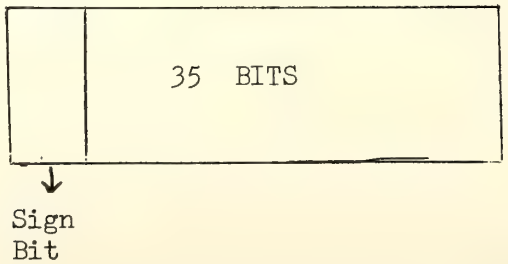
This process takes place automatically when you ask to have two numbers multiplied, but you must remember not to destroy the contents of one address by placing something else in it unless the first number is no longer needed. If you want to increment J by one, effective coding would be J=J+1, but now the old J has been destroyed. Only the new J may be referenced.

The compiler translates each instruction into numerical form, and this number is stored, along with the particular variables or constants to be acted upon, in a storage location with an address. The current address register stores the address of the instruction being performed. When one instruction is completed, the current address register is incremented by one, and the computer moves to the instruction contained in this new address.

After a program has run, it is often useful to obtain a memory dump, that is, a printing out of the contents of the computer's memory for that particular program. This printout is in the form of numbers and shows the contents of every word of the program; it thereby displays the machine-language instructions as well as the data. A dump is often quite useful in uncovering the exact location of an error. The system will automatically give an octal dump for most errors. One can take advantage of this fact and obtain a dump by creating an error condition if a dump is desired.

1.3.1. WORDS

Within the computer bits are grouped into thirty-six bit sections called words. One word is treated by the computer circuits as a unit and is transferred as such. If the word contains integer binary arithmetic data, the first bit of the word is a sign bit. A "0" indicates a positive number and a "1" indicates a negative number.



It is the words which have a name or address, and it is impossible to reference individual bits of the word except with one of the logical functions. Refer to 1.4.9.

Since binary numbers are rather hard to read and interpret, these numbers are usually converted to octal before such printouts as a dump. Therefore when the individual words are shown in a dump, there are twelve digits in length, three binary bits for each octal digit.

Furthermore these octal digits are difficult for the layman to comprehend, so another conversion must take place to print characters in the conventional decimal numbers and ordinary alphabetic characters. It must be noted that six bits are required to form one print character. There is room for six characters in each 36-bit word. It takes two octal digits to form one print character. There are several character sets in the computer world. Some require a different number of bits for each character. Print characters are often referred to as Hollerith characters (named after a former Census employee who wedded punch cards and tabulating equipment in the 1890 Census). Regardless of the character set, binary values must be converted to Hollerith characters before meaningful printing may be done. Service programs may be requested, or subroutines may be called within the user's program to accomplish conversion.

To print "off-line" (to use a printer which is not physically connected to the computer), the data must first be converted to the language which the printer will recognize. The present UNIVAC "off-line" printers require XS3 codes for printing. These printers also require special tapes (IIA) which are written with 120 words to a block. Each line that is printed uses 20 words. The format for a IIA tape is referred to as "Census compatible". The first character in a print record may be one of several symbols to control line spacing and printing in conjunction with a paper loop and a wired board. Standard boards and loops are available. In the event that special needs arise, a High Speed Printer manual is available for reference. Refer to 19.7. "UNIVAC 1107 Standard Code Translation" for a list of specific character codes. Refer to 19.12. for a list of boards and loops.

To print "on-line" (to use a printer controlled by the computer), Fieldata characters must be present. Fieldata characters are the normal 1107 High Speed Printer characters. This printer uses 128 characters for a print line; the first character is used as a print control character. An additional 4 characters are supplied by FORTRAN to meet image control requirements of the system. Therefore, records created by formatted FORTRAN "WRITE" statements for later printing consist of 22 words. The programer prepares a tape instead of using the printer directly and then requests a service routine to print from the tape. See 14.1. CENSER.

If a large volume of data is to be printed, it is recommended that IIIA tapes be generated during the processing of the program. The IIIA tapes have a higher density and are longer than IIA tapes. They hold about 8 to 10 times more data than IIA tapes. However, before

printing can be done, a service routine must be called to either copy to IIA tapes or print "on-line". See 14.1. CENSER.

1.4. NUMBER SYSTEMS

1.4.1. BINARY

For computer and automation purposes, the decimal system is difficult to use. Therefore, most computers operate using a binary or a binary coded system of number notation. To appreciate why this is so, one need only to think of some common electrical devices. Electricity and electrical implements are well suited to representing information by elements being in one of two possible states. A light can be on or off; a tube can be fully conducting or not conducting; a spot can be magnetized in one direction or another; etc. All these things may be referred to as bi-stable elements; that is they may be in one of two possible states.

For example, if we think of an ordinary light bulb, we should not have too much difficulty in reaching agreement that the bulb is either lighted or not lighted. Neither should we have trouble if we use the terms "on," or "off". These two states, "on" or "off", could be called, with prior agreement, "true" or "false", "yes" or "no", or "one" or "zero." Here then, by agreement, are two numbers or digits - "1" and "0", represented electrically. Combinations of these two digits, and only these two digits, can be used to represent numbers as we know them.

For instance, two light bulbs could represent any decimal number between zero and three:

BULBS			NUMBER REPRESENTED
Off	Off	=	Zero
Off	On	=	One
On	Off	=	Two
On	On	=	Three

Continuing in this manner, given an infinite number of light bulbs, we can represent an infinite number.

If in the diagram above, we use "0" and "1" rather than "off" and "on", it would appear as follows:

00 = zero
01 = one
10 = two
11 = three

Continuing, using one more place:

100 = Four
101 = Five
110 = Six
111 = Seven

Obviously, this is the binary number system, which is a system using only two symbols, 0 and 1.

To count in binary system follow the two rules mentioned below.

0 = Zero
1 = One (the next allowable symbol - Rule 1)
10 = Two (since no further allowable symbols exist after "1," change back to the first symbol, step one column left and advance that symbol - an implied "0." - Rule 2)

11 = Three
100 = Four (Follow Rule 2 and repeat adding the carry in the next position)
Etc.

Binary is a base two number system. (Binary digits are usually called "bits."

(The base or radix of a number may be defined as the total number of non-redundant symbols allowable in the system. Binary has a "0" and a "1," therefore, base two. Decimal - base ten.)

Since we will be using more than one system, we must distinguish between numbers of different systems. The distinction will be made by writing the number followed by the base written as a subscript, e.g.

100_{10} = one hundred
100_{2} = four

1.4.2. OCTAL

It is easier for the computer to use the binary system, but it is not so easy for us.

Suppose we have the following binary number:

001010001000010011100101110111111001

How can you express this number? It would be impractical to examine the 36 binary digits of the number to determine its value. A simplified means of interpreting and stating this binary number is needed.

If we separate the binary number into groups of 3 bits each, progressing left and right from the *binal point, and then express the value of each group of 3 bits by <u>one</u> digit we would have:

```
001 010 001 000 010 011 100 101 110 111 111 001
 1   2   1   0   2   3   4   5   6   7   7   1
```

Thus we have decreased the difficulty of expressing the binary number.

Note that in three binary digits, any value between zero and seven can be represented.

```
000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7
```

A number system which uses eight symbols, 0 through 7, is the octal system (Base eight).

The digits "8" and "9" never appear in octal.

Octal therefore, is used as a "shorthand" for binary.

Examples:

```
Binary - 010 101 100 110 110
Octal  -  2   5   4   6   6
```

$$010101100110110_2 = 25466_8$$

```
Binary - 100 000 000 000 000
Octal  -  4   0   0   0   0
```

$$100000000000000_2 = 40000_8$$

```
Binary - 001 101 111 000.100
Octal  -  1   5   7   0 . 4
```

$$001101111000.100_2 = 1570.4_8$$

The section which follows contains a brief review of some of the main points covered thus far, and then develops in detail methods for converting numbers from one system to another.

(*The term, "Binal Point," in the Binary System corresponds to the term, "Decimal Point," in the Decimal System.)

1.4.3. NUMBER CONVERSION

The 1107 computer uses the binary number system in all internal manipulations of numbers instead of the more familiar decimal system.

Of interest to our study are the methods of converting numbers from decimal to octal and binary form, and vice versa.

It is well known that the decimal system uses ten symbols to express numbers. These are the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Only two symbols are used in the binary, and eight in the octal system to represent all quantities. These are 0 and 1 in the binary, and 0, 1, 2, 3, 4, 5, 6 and 7 in the octal system.

Counting is started in the binary system in the same way as in the decimal system with "0" for zero, and "1" for one. But at two in the binary system, it is found that there are no more symbols. Thus, it is necessary to do the same thing at two in the binary system that is done at ten in the decimal system, that is, place a "1" in the first position to the left and start again with "0" in the original position. Therefore, "10" in the binary system is equivalent to "2" in the decimal system. Counting is continued in an analogous manner with a carry to the next place every time a two is reached instead of every time a ten is reached.

In the octal system counting is done in a similar fashion. The following table demonstrates how counting starts and how a few numbers are denoted in the decimal, octal, and binary systems:

DECIMAL	OCTAL	BINARY		
0	0	0	or	000
1	1	1	or	001
2	2	10	or	010
3	3	11	or	011
4	4	100	or	100
5	5	101	or	101
6	6	110	or	110
7	7	111	or	111
8	10	1000	or	001000
9	11	1001	or	001001
10	12	1010	or	001010
11	13	1011	or	001011
12	14	1100	or	001100
13	15	1101	or	001101
14	16	1110	or	001110
15	17	1111	or	001111
16	20	10000	or	010000
24	30	11000	or	011000
64	100	1000000	or	001000000
100	144	1100100	or	001100100
120	170	1111000	or	001111000

The columns in the binary and octal systems do not have the meaning of units, tens, hundreds, thousands -----, as in the decimal system. Instead, they signify units, two's, four's, eight's, sixteen's, - - - -, in the binary, and units, eight's, sixty-four's, five hundred-twelve's - - - -, in the octal system.

The bases of the decimal, octal, and binary systems are ten, eight, and two respectively. When the base of a number is not apparent from the context, it will be specified by a decimal subscript, e.g. $13_{10} = 15_8 = 1101_2$.

1.4.4. ARITHMETIC IN OTHER BASES

The algorithms for arithmetic operations are all based on properties of positional notation that are independent of the particular base in use. However, different tables apply for the additions and multiplications of digit pairs that occur in the process. Consequently, addition and multiplication tables must be reconstructed for each base concerned, and these may be used with the familiar rules for the arithmetic algorithms.

The following are addition and multiplication tables for binary and octal number systems. Where two digits occur in an entry, the leftmost is the carry digit.

Addition

Binary

	0	1
0	0	1
1	1	10

Octal

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Multiplication

Binary

	0	1
0	0	0
1	0	1

Octal

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

The following example shows the use of the addition tables and the addition algorithm to compute, in decimal, octal, and binary, the number of items in two dozen.

DECIMAL	OCTAL	BINARY
12	14	1100
+12	+14	+1100
24	30	11000

In a similar manner, the following example shows the use of multiplication to compute the number of items in a gross.

DECIMAL	OCTAL	BINARY
12	14	1100
x12	x14	x1100
24	60	0000
12	14	0000
144	220	1100
		1100
		10010000

Positional Notation

In the decimal system, a digit in a specific position within a number represents that numeric digit times 10 raised to a power determined by the number of positions away from the right most digit. The sum of these products represents the actual number. For instance, the decimal number 654321 can be considered to be the sum of the following products using the above rule:

$6 \times 10^5 = 6 \times$	100000	or 600000
$5 \times 10^4 = 5 \times$	10000	50000
$4 \times 10^3 = 4 \times$	1000	4000
$3 \times 10^2 = 3 \times$	100	300
$2 \times 10^1 = 2 \times$	10	20
$1 \times 10^0 = 1 \times$	1	1
		654321 decimal

Likewise, in the octal system the position of a digit within a number represents that numeric digit times 8 raised to a power determined by position. For example, the octal number 35276 can be considered to be the sum of the products using the base 8.

$3 \times 8^4 = 3 \times$	4096	or 12288
$5 \times 8^3 = 5 \times$	512	2560
$2 \times 8^2 = 2 \times$	64	128
$7 \times 8^1 = 7 \times$	8	56
$6 \times 8^0 = 6 \times$	1	6
		15038 decimal

Note that a number raised to a power of 0 equals 1 in any base.

In the binary system, each "1" bit represents a value which can be expressed as 2 to the power of the bit position. See 19. 9. for "Table of Powers of Two".

1.4.5. CONVERSION

It is impractical to interrogate the 36 binary digits of a number to determine its value. A simplified means of interpreting and stating this binary number is found in the octal number system.

1.4.5.1 BINARY TO OCTAL

To convert a binary number to octal, merely group the binary digits in three's, progressing left and right from the binal point. Then represent each group of 3 binary digits by 1 octal digit.

For example: 010001000111000000000010011100101110

Group the digits in three's:

010 001 000 111 000 000 000 010 011 100 101 110

Now express the value of each group of three binary digits by one octal digit:

```
010 001 000 111 000 000 000 010 011 100 101 110
 2   1   0   7   0   0   0   2   3   4   5   6
```

The maximum value of three binary bits, "111," is seven, and "7" is the highest single digit in the octal system used to represent the binary. The digits "8" and "9" never appear.

Binary numbers containing fractions: 100011001.101 are converted in a similar manner:

```
                        100 011 001  .  101
              Octal -    4   3   1       5
```

1.4.5.2. OCTAL TO BINARY

To convert from octal to binary reverse the procedure listed above. Represented each octal digit by three bits having the same value as the octal digit.

For example: Convert the following octal number to binary: 543210

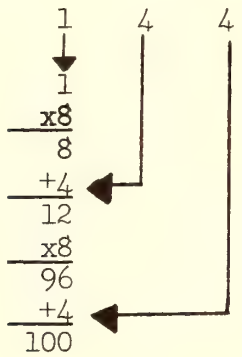
```
  5   4   3   2   1   0
101 100 011 010 001 000   Ans. 101100011010001000
```

The reason for using octal should now be apparent. Octal is a "shorthand" for binary, and is a simplified means of interpreting and expressing binary numbers. Conversion between octal and binary is immediate and can be done by inspection.

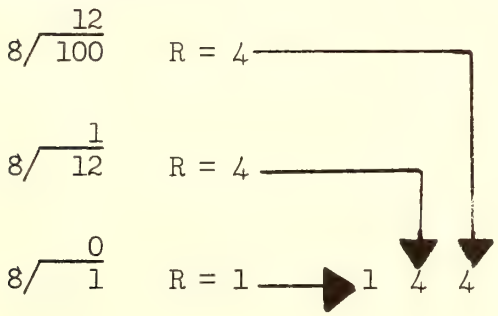
1.4.5.3. DECIMAL-OCTAL CONVERSION

Numbers can be converted from decimal to octal and vice versa by hand by using the methods outlined below. As shown, an octal number is converted to decimal by multiplying the most significant digit, adding the next significant digit, and multiplying that sum by 8. The process is continued until the least significant digit has been added to compute the decimal equivalent. Decimal to octal requires that you divide the number by eight and the remainder is the least significant octal digit. The quotient is again divided by 8, and that remainder is stored in the second least-significant place. This process is repeated until the quotient is zero.

OCTAL to DECIMAL



DECIMAL to OCTAL



1.4.5.4. BINARY-DECIMAL CONVERSION

To convert from decimal to binary, first convert to octal as previously shown, then make a direct conversion to binary by inspection. To convert from binary to decimal, reverse the procedure, by converting first to octal, then to decimal using the method previously discussed.

An alternate method of converting binary to decimal is to add the values for all "1" bits. Each bit position represents a value which can be expressed as 2 to the power of the bit position. See 19.9. for "Table of Powers of 2".

1.4.5.5. Converting Fractions

Binary to Octal

Group the bits into trio's starting from the binal point and proceed to the right. Then evaluate each trio.

$$.1010111_2 = .101\ 011\ 100$$
$$= .534_8$$

Octal to Binary

Each octal number is converted to its binary form.

$$.362_8 = 011\ 100\ 010$$

Octal to Decimal

The digits are the numerators of a series of fractions whose denominators are successive powers of 8. The sum of the fractions can then be reduced to a decimal fraction. Decimal arithmetic is used.

$$.25258 = \frac{2}{8} + \frac{5}{64} + \frac{2}{512} + \frac{5}{4096} = \frac{1365}{4096} = .333$$

Decimal to Octal

Multiply the decimal fraction by 8. The successive "overflows" beyond the extent of the original number form the octal fraction. Example: Find the octal equivalent of .1415910

$$.14159 \times 8 = 1.13272$$
$$.13272 \times 8 = 1.06176$$
$$.06176 \times 8 = 0.49408$$
$$.49408 \times 8 = 3.95264$$
$$.95264 \times 8 = 7.62112$$

The overflows produce the octal fraction .11037. Note that a terminating decimal fraction may yield a nonterminating octal fraction or vice versa.

1.4.5.6. Mixed Numbers

The integer part is treated separately from the fractional part of the number and the results are combined.

1.4.6. SIGNED BINARY NUMBERS

A binary quantity carries a positive or negative value connotation according to its representation in one's complement notation. The leftmost bit of the quantity in one's complement notation is indicative of the sign of the number represented. If the number represented is negative, the sign bit is one. A number negative in value appears in the computer as the one's complement of its absolute value. Thus, each zero in the representation of the absolute value becomes a one, and each one becomes a zero. For example, the representation of the positive decimal quantity 13 as a computer word is:

000 000 000 000 000 000 000 000 000 000 001 101

The representation of the negative decimal quantity 13 is the one's complement of this, or:

111 111 111 111 111 111 111 111 111 111 110 010

Note: The 36 bits of a computer word are grouped in threes as above to promote ease of reading and converting to octal notation.

The octal representation of the positive decimal quantity 13 is:

000000000015

The octal representation of the negative decimal quantity 13 is:

777777777762

Any bits between the sign bit and the most significant bit of the number represented are also indicative of the sign of the number. Thus the sign bit is duplicated to the right until the most significant bit appears the same as the sign bit. Note also that the most significant bit of a quantity always differs from the sign bit. This is less obvious when a number is represented in octal notation.

The quantity "zero" has a positive and a negative representation in the computer, namely, in octal notation, 000000000000 or 777777777777. However, it is not possible for the representation of "negative" zero, 77777777777, to be generated as the result of an arithmetic operation. A "negative" zero generated in some other way, is treated as the quantity zero in arithmetic operations.

In an n-bit register, no more than n-1 bits can represent the value of a signed integer since at least one bit represents the sign. Thus, the range of signed integers which can be represented in an n-bit register is minus $(2^{n-1}-1)$ to plus $(2n^{-1}-1)$.

Hence, the positive integers which can be represented in a 36-bit register lies in the range of 1 to $2^{35}-1$, or as expressed in octal notation

000000000001 through 377777777777

The negative integers which can appear in a 36-bit register lie in the range -1 to $-(2^{35}-1)$, or as expressed in octal notation

777777777776 through 400000000000

The range in decimal would be

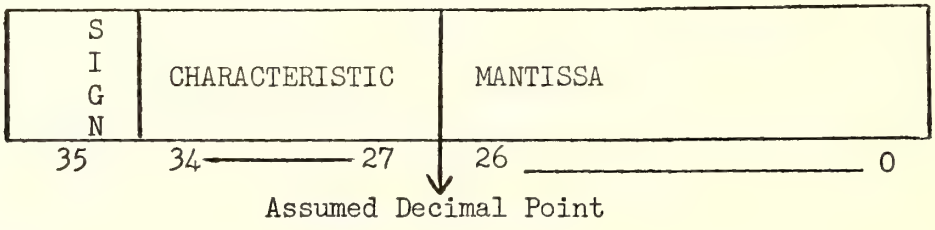
from -34,359,738,367 through +34,359,738,367

Examples of signed binary numbers (expressed in octal) are:

Octal	Decimal Value
000000000144	+100
777777777633	-100
000000000001	+1
777777777776	-1

1.4.7. FLOATING POINT

The binary representation of a real number is sometimes referred to as a floating point number because of the scaling which occurs as the number is packed into the form shown below:



The characteristic (an 8-bit field) represents the scaling of the number. A number as large as 255 can be contained in 8 bits. There is a need to provide for negative characteristics, so half of 255 + 1 is 128. Characteristics may now range from -128 to +127. Because there is no sign bit for the characteristic, the numbers are biased by 128. Thus, all numbers of the characteristic become positive. The number -128 now becomes 0; the number 0 becomes 128; and the number +127 becomes 255. Expressing this in octal: negative characteristics range from 0-177, and positive characteristics range from 200-377.

The mantissa is fractional and therefore is always less than 1. The sign bit in bit 35 determines its sign. When scaling of an integer takes place to create the mantissa, the number of places necessary to move a number from right to left to place the most significant bit at bit position 26 is subtracted from the number 155.

Thus if the original number is 1, it would have to be moved 26 places to the left to read the assumed decimal point between bit 27 and 26. Subtracting 26 from 155 gives 129. This in turn becomes octal 201, the characteristic for this number. The octal floating point number for 1 would then be this: 201400000000. The octal floating point number for -1 would be 576377777777 since a negative number is expressed as the complement of the positive number.

Floating point is a solution to the problem of keeping track of decimal points by representing every real number as a decimal fraction between .1 and 1.0 multiplied by a power of 10. Thus:

$$.0028 = .28 \times 10^{-2}$$

$$108.75 = .10875 \times 10^{3}$$

In other words, the characteristic is the exponent and the mantissa is the fraction.

1.4.8. BINARY CODED DECIMAL CODES

The digital computer can be thought of as an assemblage of two-state devices because it manipulates the one's and zero's of the binary number system. People, on the other hand, are more accustomed to decimal numbers, and for this reason it is often desirable to build a computing system which can be operated in decimal.

To build a decimal computer with two-state devices, it is necessary to encode the decimal digits with binary bits. Four binary bits are needed. Although only 1Ø of the 16 permutations possible with the 4-bit decade will be used, all are available.

The choice of code is obviously important. Desirable features of the code are: Ease in performing arithmetic operation, economy of storage space, error detection and correction, and simplicity. Several possible codes are shown below.

DECIMAL	8421	EXCESS 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	0110
8	1000	1011
9	1001	1100

Four Bit Codes

The 8421 code is commonly referred to simply as binary-coded decimal because the weights of the positions are the same as in the binary number system. Arithmetic operations are easily performed using the same basic method as in binary since the number sequence is the same.

In the Excess-3 code, a decimal number D is represented by the binary equivalent of the number D+3. The Excess-3 code is not a weighted code, but since it follows the same number sequence as binary, it is useful in arithmetic operations. Addition is facilitated since the need for a correction factor is easily detected and easily implemented. Because it is self-complementing, the Excess-3 code is also useful in subtraction.

1.4.9. Logical Operations

Certain operations which we refer to as logical, or Boolean, allow bits to be extracted or changed depending upon corresponding bits in another word (a mask). This allows us to set, clear, or to switch bits. The "exclusive or" combination of binary digits is when one of the digits, but not both, is one. It is found by adding two bits and disregarding any carry to the next bit position. The rules are as follows:

0	1	0	1
0	0	1	1
0	1	1	0

The logical sum, "inclusive or", is when either, or both, of the binary digits is one. The truth table is as follows:

0	1	0	1
0	0	1	1
0	1	1	1

The logical product, "and" or "conjunction", is when both of the binary digits are one. It is found by multiplying two bits. The results of an "and" operation are as follows:

0	1	0	1
0	0	1	1
0	0	0	1

1.5 THE 1107 CENTRAL COMPUTER

The 1107 Central Computer is composed of the same basic elements as other computers, namely control unit, arithmetic logic, storage, and input/output elements.

1.5.1. The Control Portion

The control portion of the computer determines which operations are to be executed and sets up the sequence of events that takes place in the execution of a given instruction. The central computer has a repertoire of 117 instructions, which can perform such operations as transfers, arithmetic tests, and shifts. The 1107 operates on single address logic. Each instruction only specifies one operand even though other registers (such as an accumulator and/or an index register) are involved in the execution. This results in the programmer actually coding more instructions than with a machine using two-address logic. One feature of the 1107 that works to offset the number of instructions required for a program is the large number of special registers available. It has 16 arithmetic registers, 15 index registers, (which are used to increment or decrement operand addresses), and several other special registers.

One feature of the 1107 that is valuable in cutting the time it takes to execute instructions is the logical division of core memory into two separate banks. Whenever the contents of a core memory location are read, the contents are destroyed. Part of the necessary cycle of operation is restoration of the location that has been referenced. The reading and restoring of the contents of a register comprises the read-restore cycle. Thus, when an instruction is brought up for execution, one read-restore cycle is required. If an operand is involved, another read-restore cycle is required. If however, instructions are stored in one bank of core and operands are stored in the other bank it becomes possible to cut the cycle time in half. While a register in one bank is being read, a register in the other bank can be restored.

1.5.2. The Computer Memory

The computer memory of the 1107 is of two types: Magnetic core and thin film. The Census 1107 configuration has 65,536 core storage registers. A feature of the 1107 is thin film memory. This film memory consists of very thin spots of metallic oxide that have been deposited on glass. Thin film has a much more rapid rate of access than core memory. There are 128 thin film registers in the 1107. These are the special registers already mentioned. Their access speed is such that thin film may be referenced six times during one reference to core memory.

1.5.3. The Arithmetic Section

The arithmetic section of the 1107 handles arithmetic and comparative functions. The addition and subtraction functions can be carried out in several ways. In addition to a straight addition and subtraction there are instructions to add or subtract magnitude. Operations using partial words may also be specified. For example, you can add any third of a word to the contents of an Arithmetic Register, or subtract any sixth of a word. This partial word facility is not limited to arithmetic instructions. These are not the same as partial word arithmetic instructions. For example, you can simultaneously add or subtract all the thirds of a word to or from all the thirds of an A register.

Along with the addition and subtraction are multiply and divide instructions. Multiplication or division can specify either 36 or 72 bit results. The number of A registers available makes coding of complex arithmetic operation less involved, since intermediate results can be left in A registers.

The 1107 has a great number of comparative or test instructions. In most cases these test instructions come in pairs; that is, for each test of one mutually exclusive condition there is a test for its opposite. For example, there is a test for zero and a test for not zero. There are instructions that will test within limits or test outside limits. There are also a series of instructions which will search a specified number of locations, testing each until a find is obtained. These search instructions

are also available with masking, so that a test can be made against only the extracted bits of a number of words.

Floating point arithmetic is also part of the repertoire of the 1107.

1.5.4. Input-Output Channels

The 1107 has multiple channels for input-output, each one of which can be used for data input or output. Furthermore, each channel can have a number of units associated with it. For example, you might have six servos on one channel, two on another, and a drum on a third channel. You could only read one unit on any channel at a time, but you could be reading on all three channels concurrently, or reading on two and writing on one, etc. Only one word can be stored in or written from the central computer at a time, but the central computer is so fast that it can handle one write on tape, read one from tape, and then come and get the next word from drum while still executing program instructions. All this with the drum going full speed and utilizing interlace.

Each channel has its own channel synchronizer and control unit as well as whatever units are attached to that channel. The channel synchronizer handles communication to and from the central computer, including transfer of data words. The control unit controls the operation of the associated peripheral units.

1.5.5. Peripheral Equipment

The peripheral equipment that is associated with the Census 1107 is only part of what could be associated. One item of peripheral equipment is the drum. Note that the drum is not part of the central computer memory and cannot execute internal processing instructions. The drum is a storage device. The Census configuration has one drum subsystem with two drum units, and provides more than 500K memory locations of 36 bits on each unit. After the initial access time required to find the first location the drum transfer rate is 16.5 M/seconds per word. The use of the drum as storage for program segments or tables during a run is thus very feasible.

The FASTRAND II Subsystem also provides a storage medium for programs and their source data. The subsystem is composed of three FASTRAND II drums with a storage capacity of 22 million words each.

There are three types of Servos that are associated with the Census 1107. One type, the Uniservo IIA has variable recording density of 125 and 250 characters to the inch, and thus a variable transfer rate of 12.5 or 25,000 char. per. sec.

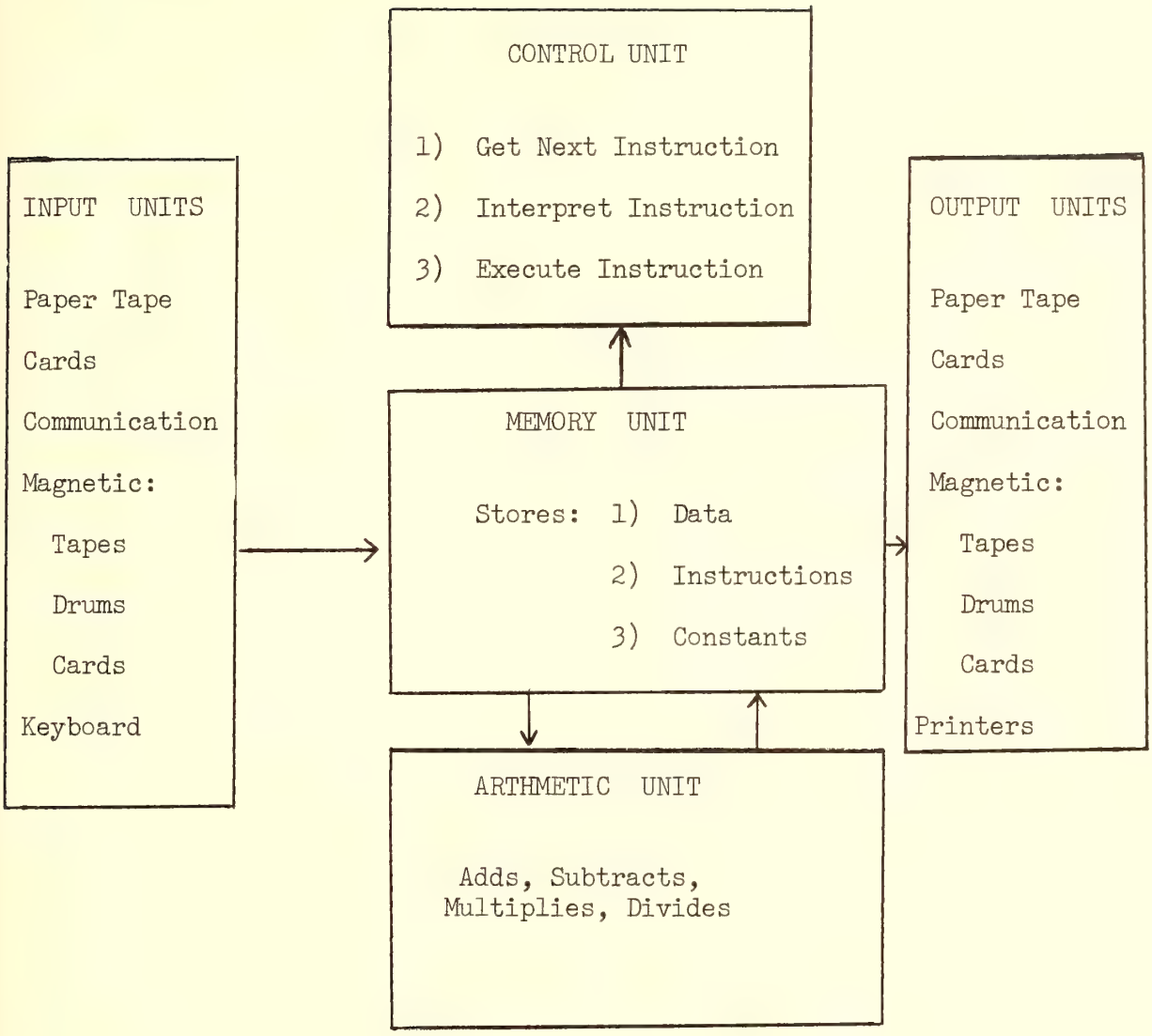
The Uniservo IIIA's are a new model Servo with a different technical principle than the IIA's. These servo's have a

transfer rate of 120K char. per. sec. and a rewind speed of 300 inches per second. With a dual channel synchronizer it is possible to read and write or to read two units simultaneously. The magnetic tape used with this servo is physically different from the tape on the Uniservo IIA. The High Speed Printer is different from the off-line HSP's and it is on-line. This does not mean that printing ties up the computer, since printing can take place concurrently with a running program. A card-reader is also included as on-line equipment.

The Uniservo IVC's are a magnetic tape subsystem which allows complete compatibility between the Univac 1107, and IBM data-handling equipment with off-line tape conversion. The IVC tapes can be read or written with a density of either 200, 556, or 800 characters per inch.

The Census configuration has a Paper Tape Reader and Paper Tape Punch. The paper tape reader can read at a rate of 400 c.p.s. The punch can punch up to 110 frames p.s.

The Supervisory Console Printer provides facilities for programed typeouts from the Central Computer. It also provides for type-ins of required information or parameters. The console is equipped with 15 console selective jump switches which are available to programs, as well as 4 selective stops. However, FORTRAN programers do not use these switches.


CONTROL UNIT
1) Get Next Instruction
2) Interpret Instruction
3) Execute Instruction
INPUT UNITS
Paper Tape
Cards
Communication
Magnetic:
Tapes
Drums
Cards
Keyboard
MEMORY UNIT
Stores: 1) Data
2) Instructions
3) Constants
OUTPUT UNITS
Paper Tape
Cards
Communication
Magnetic:
Tapes
Drums
Cards
Printers
ARTHMETIC UNIT
Adds, Subtracts,
Multiplies, Divides

1.6. EXERCISES:

1.6.1. Define the following:

a. bits

b. sign bit

c. word

d. printer character

e. software

1.6.2. There are _____ bits to a word. _____ bits make one octal character. _____ octal characters make a word. Each printer character is _____ octal characters. There are _____ printer characters to a word.

1.6.3. The decimal value of: 777777777712_8 is:

000000000025_8 is:

777777777623_8 is:

000000000123_8 is:

777777777765_8 is:

1.6.4. Show the octal contents of the word generated for each of the following values.

a. -1

b. 200_{10}

c. 65_{10}

d. -65_{10}

1.6.5. If you wanted to print the number "6219," could you just generate the value 6219?

If you answered NO, what would you generate?

1.6.6. Circle the positive values:

a. 177777777711_8

b. 421321421321_8

c. 700000777734_8

d. 345673456712_8

e. 333333322222_8

f. 123456765432_8

g. 537117355371_8

h. 223344556677_8

i. 377777777777_8

j. 400000000000_8

1.6.7. Show the complement for each of the above.

1.6.8. Perform the following operations:

a. $101001_2 + 110110_2$

b. $111111_2 + 10100_2$

c. $101101_2 - 110_2$

d. $110110_2 - 111_2$

e. $110110_2 \times 111_2$

f. $10101100_2 \times 10100_2$

g. $76532_8 + 7321_8$

h. $7134\emptyset5_8 + 6616_8$

i. $137362_8 - 41414_8$

j. $217654_8 - 24566_8$

1.6.9. Convert to decimal:

a. 1654_8 b. 50131_8

c. 3162_8 d. 12345_8

e. 427_8

1.6.10. Convert to octal:

a. $10110110111010101011001_2$ b. $11110001001100100010 1_2$

c. $00011011010101011011 0_2$ d. $11100001000100101111 0_2$

e. 931507_{10} f. 18902_{10} g. 2135_{10}

ANSWERS TO EXERCISES

1.6.1. a. A bit is the basic unit of information with which a computer works. It may have a value of 0 or 1.

b. A sign bit is the first bit of a word.

c. A word is a group of 36 bits.

d. A printer character is a group of six binary bits.

e. Software is the program of instructions that puts the system to work.

1.6.2. 36, 3, 12, 2, 6

1.6.3. -53, 21, -108, 83, -10

1.6.4. a. 777777777776 (-1)

b. 000000000310 (200)

c. 000000000101 (65)

d. 777777777676 (-65)

1.6.5. The value 6219 would need to be expressed as Hollerith characters. (A way will be shown later whereby a constant may be expressed as consisting of Hollerith characters in a form suitable for printing.)

1.6.6. a., d., e., f., h., i.

1.6.7. a. 600000000066

b. 356456356456

c. 077777000043

d. 432104321065

e. 444444455555

f. 654321012345

g. 240660422406

h. 554433221100

i. 400000000000

j. 377777777777

1.6.8. a. 1011111 base 2
b. 1010011 base 2
c. 100111 base 2
d. 101111 base 2
e. 101111010 base 2
f. 110101110000 base 2
g. 106053_8
h. 722223_8
i. 75746_8
j. 173066_8

1.6.9. a. 940 b. 20569
c. 1650 d. 5349
e. 279

1.6.10. a. 5556531_8 b. 7423105_8
c. 0665266_8 d. 7021136_8
e. 3433263_8 f. 44726_8 g. 4127_8

2. THE FORTRAN LANGUAGE FUNDAMENTALS

2.1. WRITING FORTRAN PROGRAMS

A FORTRAN (FORmula TRANslator) statement may be written anywhere in columns 7-72 of the FORTRAN coding sheet. Except in special cases, blanks are ignored by the FORTRAN processor and may be used to improve the readability of the source program printout. If the statement is too long to be contained on a single line, it may be continued on up to 19 successive lines by placing any character, other than zero, in column 6. Columns 1-5 may be used to write statement labels which may be any number from 1 to 32767. These labels are used when referencing a statement. These labels need not be in any order. Columns 73-80 are ignored by the processor, but may contain any information which will be helpful to the programer. Entire lines will be ignored by the processor provided that column one (1) contains a C. In this manner, comments may be included within the program or blank lines may be used to separate sections of the program.

COLUMNS	PURPOSE
1	C (comment)
1-5	Statement label (numbers 1-32767)
6	Any character besides zero or blank indicates the line is a continuation of previous line.
7-72	FORTRAN statement
73-80	Any helpful information

FORTRAN has no "lower case" letters, therefore only "CAPITAL" letters should be used when coding.

2.1.1. BASIC CHARACTERS

Only the characters in the following lists can be used in the structure of the FORTRAN language:

letters: A,B,C,D,E,F,G,H, I,J,K,L,M,N,O,P,Q,R,S,T,U, V,W,X,Y,Z

digits: 0,1,2,3,4,5,6,7,8,9

arithmetic operations: +,-,*,/,**

relational operations: .GT.,.GE.,.LT.,.LE.,.EQ.,.NE.

logical operations: .NOT.,.AND.,.OR.

2.1.1. BASIC CHARACTERS (Continued)

logical values: .TRUE.,.FALSE.

delimiters: (,),/,.

special: $

assignment: =

2.1.2. PREPARATION OF PROGRAMS AND RELATED MATERIALS FOR KEY PUNCHING

In order to avoid misunderstanding and resultant key punching errors, please use the following notations for certain characters:

WRITTEN AS	INTERPRETATION	COMMENT
0 or Ø	0 (Numeric)	Slash optional but preferred.
O	o (Alpha)	Must have underline.
/	1 (Numeric)	Do not use flag and/or bottom bar.
I	i (Alpha)	Use bar at top and bottom as in capital "I".
2	2 (Numeric)	Do not slant bottom upwards.
Z	z (Alpha)	Add bar thru center of "Z".
5	5 (Numeric)	Do not underline.
S	s (Alpha)	Must have underline.
J	j (Alpha)	Must have bar at top and underline.
U or u	u (Alpha)	Downstroke to "u" preferred. Do NOT UNDERLINE.
V	v (Alpha)	Must have underline.
/	slash	From corner to corner.

FORM BC-426
(2-11-65)

FORTRAN CODING

PROBLEM

C	STATEMENT NUMBER	CONT.	STATEMENT
1	2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
C		G	RAM TO FIND THE AREAS OF TRIANGLES
C	THE		LENGTHS OF THE SIDES ARE READ IN
C			FROM A CARD-TO-TAPE REEL TO A, B, C
C			
	20		READ (5,23) A,B,C
	23		FORMAT (3F10.0)
			S=(A + B + C) / 2.0
			AREA=SQRT(S*(S-A)*(S-B)*(S-C))
	25		WRITE(6,17) A,B,C,AREA
	17		FORMAT (1P4E16.7)
			GO TO 20
			END

2.1.3. SOURCE STATEMENTS

The job of the programer is to compose a series of instructions which become the source program. When the computer performs these instructions in the specified order, the data is processed to obtain the desired results. Each instruction is concerned with one aspect of the problem; it may cause data to be fed into the computer, calculations to be performed, decisions to be made, results to be printed, etc. Some statements merely provide information to the processor at assembly time. The processor then expands this source program into a language which the computer can understand and these object program (machine language) instructions are fulfilled.

Executable Statements

Arithmetic	CALL
Logical	RETURN
Unconditional GO TO	STOP
Assigned GO TO	READ
Computed GO TO	WRITE
ASSIGN	PRINT
Arithmetic IF	PUNCH
Logical IF	REWIND
DO	BACKSPACE
CONTINUE	END FILE

Non-executable Statements

SUBROUTINE	DOUBLE PRECISION
FUNCTION	COMPLEX
FORMAT	LOGICAL
PARAMETER	EXTERNAL
INTEGER	ABNORMAL
REAL	END

2.2. CONSTANTS

A constant is any quantity which appears in a FORTRAN statement in the form of a specific numeric or logical value.

Every constant is interpreted as a binary number by the processor, since the computer can interpret only two signals -- charge and no charge. The memory of the computer consists of a series of words which may be referenced. Each word contains 36 bits (binary digits), each of which is either a charge or no charge.

1. No commas may be written within a constant.

2. All constants (except logical)

   a. if negative, must have a minus sign
   b. if positive, may have a plus sign
   c. if no sign, the constant is assumed to be positive.

There are five types of constants:

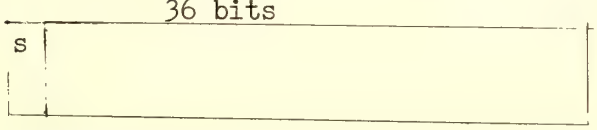
1. Integer
2. Real
3. Double Precision
4. Complex
5. Logical

In addition, a string of Hollerith characters preceded by wH may be used as a constant where "w" stands for the number of fieldata characters. If "w" is not divisible by 6, the field will be space filled on the right. See 7.5. and 5.4.5.9.

2.2.1. INTEGER - FIXED POINT

An integer is a whole number without a decimal point. It may be positive or negative. The absolute value of the number may range from 0 to 34359738367 = $(2^{35}-1)$ (1 to 11 digits).

The integer number is represented in the computer as a binary number which occupies one full word of 36 bits. The first bit of the binary word is the sign (0 for positive, 1 for negative).



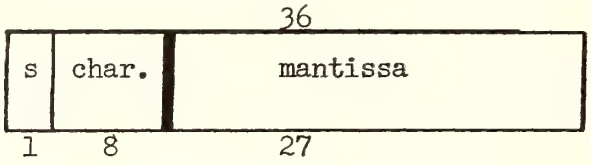
Examples of INTEGER numbers:

```
     0
    12
  1234
     6
 10000
.12345678
```

2.2.2. REAL - Single Precision Floating Point

A real number may be a whole number, a fractional number, or a mixed number with a decimal point. There may be from 1-9 significant digits. The magnitude must be in the range 10^{-38} to 10^{+38}. The decimal point may express the exponent, or a power of ten may be applied if the number is followed by the letter "E" with a signed (+ is optional) one or two-digit exponent.

The real number occupies one computer word in hardware single precision floating point format which consists of a sign bit, an 8-bit characteristic (octal 200 bias) and a 27-bit mantissa.



Examples of REAL numbers:

FORTRAN NOTATION	ACTUAL NUMBER
.1E3	100
2.56	2.56
-.43E5	-43,000
1E12	1000000000000
-.492E-4	-.0000492
.123456E2	12.3456
1.	1
-.0000492	-.0000492
-43000.	-43,000
12.3456	12.3456

2.2.3. DOUBLE PRECISION - Double-Precision Floating Point

A double precision number may be a whole number, a fractional number, or a mixed number with a decimal point. There may be from 1 to 17 significant positive or negative digits. The magnitude must be in the range 10^{-38} to 10^{+38}.

A double precision constant must be followed by the letter "D" with a signed (+ is optional) one or two-digit exponent.

The double precision number occupies two consecutive storage locations. In the 1107, with the sign bit being in the first bit of the first word, the format is that of two real numbers. However, the 1108 hardware recognizes a different format.

Examples of DOUBLE PRECISION constants:

FORTRAN NOTATION	ACTUAL
0.0D0	0
1.0D0	1
16.9D+1	169
+8.897D-10	.0000000008897
-1750.D+3	-1750000
123.4567891D0	123.4567891
.1234567891D3	123.4567891
16D0	16

2.2.4. COMPLEX

A complex number is written in the form (r,i) where "r" represents the real part and "i" represents the imaginary part. Each part is internally represented by one single precision floating point word. Each part is written as if it were a real constant. The two parts occupy consecutive locations in memory. The rules which apply to REAL constants apply to each part.

Examples of acceptable COMPLEX constants:

FORTRAN Notation

(0.0,1.0)	1.0i (pure imaginary)
(3426.78,-293.6)	3426.78-293.6i
(4.12E2,6.5)	412.+6.5i
(4.12E-2,6.5E3)	.0412 + 6500i
(4.12E-10,6.5E-3)	.000000000412+.0065i

2.2.5. LOGICAL

There are two logical constants: .TRUE. and .FALSE.

The periods are part of the logical value notation and must be written.

The internal representation is:

If .TRUE., the right most bit is a 1.

If .FALSE., the right most bit is a 0.

2.3. VARIABLES

A variable is a named position in memory which will contain a single value at any one time, but this value may be changed an infinite number of times throughout the program; but for any given time, each variable has only one value. It may be referenced by using its symbolic name in any way that a constant can be used. The old value will be destroyed when a new value is placed within this position. The symbolic name is composed of from one to six alphanumeric characters, the first of which must be alphabetic. No special symbols are allowed. The assignment of names is under the control of the programer. There are five types of variables: INTEGER, REAL, DOUBLE PRECISION, COMPLEX and LOGICAL. The type is specified either implicitly

2.3. VARIABLES (Continued)

or explicitly at the beginning of the program. A variable will occupy the same number of storage locations as a constant of the same type and may take on any value its corresponding type of constant may assume.

Specifying the Variable TYPE

If the variable name begins with the letters I-N, the variable will implicitly be defined as "integer" type. If the variable name begins with any other letters, the variable will implicitly be defined as "real" type. These rules may be overridden by the use of type statements at the beginning of the program. For example, INTEGER A will cause A to be an integer variable, whereas it would have been "real" without this statement.

Double precision, complex, and logical variables must be defined in a type statement. The format for the type statement is TYPE A,B,C where TYPE is one of the five types of variables and A,B, and C are the variables, separated by commas, which are to be that type.

Examples:

```
REAL  I,J,K

INTEGER  A,B,C,D

DOUBLE PRECISION  N,D,X

COMPLEX  M,V,Y

LOGICAL  L,Q
```

Examples of acceptable implicit variable names:

REAL	INTEGER
G	I
GAMMA	J
G1	K
R19	IJK
X	LARGE
SIGMA	N2
	N
	IBM
	LAST
	NG

2.4 OPERATORS

There are five basic arithmetic operations--addition, subtraction, multiplication, division, and exponentiation. Each is represented by a distinct symbol.

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	** (considered to be one symbol)

It is never correct to write two operators side by side. These 5 symbols are the only ones used in a FORTRAN statement. Any others must be composed of the basic five or computed by using the mathematical functions that are discussed later.

2.5. EXPRESSIONS

A FORTRAN expression is a rule for computing a numerical value. An expression may consist of a single constant, a single variable, or a single function reference (described later). These elements may be combined by using operation symbols and parentheses to build up more complex expressions.

The rules for writing expressions are:

1. Two operation symbols must not appear next to each other. Thus A*-B is incorrect. However, A* (-B) is correct, or -B*A is correct.

2. Parentheses do not indicate multiplication, but indicate that the operations within the innermost parentheses are to be performed first.

3. When the order of operations is not specified by the use of parentheses, the sequence is as follows: All exponentiations are performed first, then all multiplications and divisions from left to right, and finally all additions and subtractions from left to right.

   Thus, these two expressions achieve the same effect:

   a. A*B+C/D-E**F

   b. (A*B)+(C/D)-(E**F)

4. Parentheses must be used to indicate groupings just as in ordinary mathematical notation. Parentheses force the inner operation to be done first. Thus $(X+Y)^3$ must be written (X+Y)**3 to convey the proper meaning. X+Y**3 would change the meaning, although it is a valid expression.

2.5. EXPRESSIONS (Continued)

5. The ambiguous expression A^{B^C} must be written as A**(B**C) or (A**B)**C, whichever is intended, but never as A**B**C.

6. Integer expressions may be raised to an integer power. Real expressions may be raised to either an integer or a real power.

7. If the terms of an expression are of varied type, the priority of the result is DOUBLE PRECISION, REAL, INTEGER. For example, if any term of an expression is DOUBLE PRECISION, the answer will be DOUBLE PRECISION. Likewise, if (and only if) all the terms are integers will the result be an integer.

8. If one of the terms of an expression is complex while another is real, the operation will be performed as if all the terms are complex. The real numbers will be assumed to have an imaginary part of zero. Necessarily, the result will be complex.

   It is illegal to have a complex and a double precision number within one expression since the result would be too large to fit into the allocated space.

9. Complex data can be used in an expression with other complex data or with real data. In operations not involving exponentiation, the mixing of a real and a complex quantity results in a complex value. However, raising a complex value to a real power affects both the real and imaginary parts.

10. Logical data can form an expression only with other logical expressions. See 2.8.

Examples of Expressions

```
1
A+B
I
X**1.67
4
7+K*4
B**2-4.*A*C
3.
X/4
ABC+DE7*UE7
(1.+S/T)**(T*P)/R
A*B*C**2
(C/A*D)**2
(A+B)+(C+D)
A**3+D
D*(A+B)
(B+C+A+B)**D
A/B/(C*D**A)
X+Y**3
A+B/C
A+B/(C+D)
((A+B)/(C+D))**2+X**2
(X+2.0)/(Y+4.0)
2*N+1
A.GE.B
A.NOT..GT.B
```

2.5.1. INTEGER DIVISION

Any fractional portion resulting from the division of integer data is truncated without rounding. The following examples illustrate FORTRAN integer division:

5/4 produces a result of 1

1/2 produces a result of 0

10/4 produces a result of 2

But in real mode

5.0/4.0 will give the result 1.25

4.0/5.0 will give the result .8

The MOD function is available to return the remainder as the value of the function when integer division is performed. The AMOD function is used for real division. An example of the MOD function would be to divide integer N1 by integer N2 and save the remainder in N3:

```
N3 = MOD(N1,N2)
```

Successive integer divisions should be clarified by using parentheses. The following illustrate how the use of parentheses may produce different results:

Expression:	(I/J)/K	I/(J/K)
Step 1.	(4/3)/2	4/(3/2)
Step 2.	1/2	4/1
Step 3.	0	4

2.5.2. LIMITATIONS OF ARITHMETIC OPERATIONS

The programer is reminded that a real value is an approximation to a real number, and any arithmetic operations involving real values may introduce an error in the least significant digit of the result due to rounding and truncation.

During computations, certain faults (characteristic underflow, characteristic overflow and divide overflow) may sometimes occur. When a fault of this nature occurs, a result of zero is returned. The programer may call the Census subroutine CHKDIV, which returns control to the programer's coding for special action; or he may call NOCHK to terminate the run with a memory dump. (See 9.1. and 9.2.)

It should be noted that division by zero results in divide overflow.

2.5.3. FUNCTION REFERENCE

A function reference is evaluated prior to that point where its value is required as a term. Thus, in the expression A**SIN(B), the sine of B is found before exponentiation is performed. In the expression A**SIN(X+Y), the sum of X+Y is found before the sine is evaluated.

2.6 ARITHMETIC ASSIGNMENT STATEMENTS

An arithmetic assignment statement is an order to FORTRAN to compute the value of an expression on the right and to give that value to the variable named on the left. The equal sign is that order. All of the arithmetic necessary to create a value is done on the right side of the equal sign. That value is then placed into the variable designated on the left side of the equal sign.

Examples:

```
N = N+1

R = (A+B*X)/(C+D*X)

DELTA = BETA+2

W = (A-B)**2

C = A*A+B*B

R = R**2

P = ((F+G)/(R+S))**2

G = G+2

A = -1.0*A

D = 1.1*D
```

2.7. ARITHMETIC IF

The arithmetic IF statement determines the statement to be executed next, dependent upon the value of an arithmetic expression.

The general form of the arithmetic IF is as follows:

IF (arithmetic expression) $label_1$, $label_2$, $label_3$.

The arithmetic expression may not contain complex values. It must be enclosed in parentheses.

This statement provides for a conditional three-way branch. If the value of the arithmetic expression is negative, control is transferred to the statement specified by $label_1$. If the value is zero, control is transferred to the statement specified by $label_2$. If the value is positive, control is transferred to the statement specified by $label_3$.

Note that the label may be a switch variable to which a statement label was previously assigned by an ASSIGN statement. See 2.13.2.

2.7. ARITHMETIC IF (Continued)

A convenient feature of the Census FORTRAN is the omission of a label if it is the intent to allow control to fall through to the next statement.

Examples of Use:

```
IF (NUMBER) 5,10,5

IF (J-6) ,  , 50
```

2.8. LOGICAL EXPRESSIONS

A logical expression is one which takes on the value .TRUE. or .FALSE..

A logical constant is a logical expression.

A logical variable or function is a logical expression.

The relational operators used in logical expressions are:

.EQ.	Equal to
.NE.	Not equal to
.LT.	Less than
.LE.	Less than or equal
.GE.	Greater than or equal
.GT.	Greater than
.NOT.	Used prior to expressions including these relational operators it reverses the logic
.AND.	True if both conditions are true
.OR.	True if either condition is true

Logical variables may not be used with the relational operators except .AND. and .OR..

As in arithmetic operations, when the order of operations is not specified by the use of parentheses, .LT., .LE., .EQ., .NE., .GT., .GE. are evaluated first, followed by .NOT., then .AND. and finally .OR..

2.8. LOGICAL EXPRESSIONS (Continued)

Examples of Logical Expressions:

```
A+1 .GT.B.OR.C.AND.G+2.NE.H

X.LE.2.1

X.GE.EPS.AND.NUMBER.LT.10

.NOT.(X.LT.12.0)
```

2.9. LOGICAL STATEMENTS

The logical statement has the form v = e where v is a logical variable and e is a logical expression.

Rules:

1. The logical variable will be .TRUE. if the expression is true.
2. The logical variable will be .FALSE. if the expression is false.

Examples:

```
LOGICAL  A,B,C,D,E

A = N.EQ.M

B = N.EQ.10

C = X.LT.12.0

D = .NOT.(X.LT.12.0)

E = X+1.GT. Y.OR.Z.AND.G+2.NE.H
```

2.10. LOGICAL IF

The logical IF statement has the general form IF (e)S where e is a logical expression and S is any other statement except another logical IF or a DO. If the logical expression is true, then the statement S is executed. If the logical expression is false, control will be transferred to the next executable statement following the logical IF. See chapter 4 for a discussion of the DO statement.

Examples:

```
IF (A.GT.B.AND.C.LT.D) GO TO 50
GO TO 28
IF (N.EQ.L .AND.M.EQ.20) N=N+1
```

2.10. LOGICAL IF (Continued)

Whenever an integer constant is used within a Logical IF expression, then that constant represents a decimal number. Sometimes it is desired to extract an XS3 character from a record and compare the character to a decimal number. In order to select the right constant, the XS3 character must be converted to octal and the octal should then be expressed as a decimal integer. For example, it is desired to check for the XS3 character "7." Checking the code translation table, one sees that XS3 "7" is really octal "12". Converting octal "12" to decimal, the correct integer constant is "10". A table of XS3 numbers appears below:

XS3 Char	Octal	Decimal
0	3	3
1	4	4
2	5	5
3	6	6
4	7	7
5	10	8
6	11	9
7	12	10
8	13	11
9	14	12

Likewise, a table of Fieldata characters appears below:

Fieldata Character	Octal	Decimal
0	60	48
1	61	49
2	62	50
3	63	51
4	64	52
5	65	53
6	66	54
7	67	55
8	70	56
9	71	57

Any Hollerith character may be expressed as a decimal number. Consult the language code list for the octal code of the character.

2.11. PARAMETER VARIABLES

It is sometimes desired to compile a program several times where the values of certain INTEGER constants are the only changes of the FORTRAN text from one compilation to the other. Instead of changing these constants throughout the program each time, a symbolic name may be used which is to appear in a PARAMETER statement before any other reference to the name:

$$\text{PARAMETER } i_1 = m_1 \ , \ i_2 = m_2 \ , \ . \ . \ . \ . \ . \ , \ i_k = m_k$$

where the "i's" are any symbolic names (See 2.3) and the "m's" are INTEGER constants. The effect of referencing "i" is exactly the same as if the INTEGER constant "m" would stand in its place.

This symbolic name, called PARAMETER variable, therefore must not appear in places where INTEGER constants cannot appear (e.g. as left hand variable of an arithmetic statement, etc.).

Example of a PARAMETER statement:

```
PARAMETER J=1, K2=-4, ZEST=654
```

The use of PARAMETER variables is most useful in achieving variable dimensions in arrays from one assembly to another. Refer to chapters 3 and 7.

Example:

```
PARAMETER  N= 100, M = 200

DIMENSION  X(N), T(M), Z(N,M)
```

2.12. CONTINUE STATEMENT

The statement

```
CONTINUE
```

itself causes no action in the object program, but provides a valuable means to terminate an iteration of a DO-loop skipping the last statement (s) of the range. A control statement any place in the middle of the range deciding the termination of the iteration transfers control to a CONTINUE statement. The CONTINUE statement must be the last statement in the range; that is, the statement number preceding it also must appear in the DO statement. If the iterations of an inner DO-loop are terminated as described above, the CONTINUE statement cannot be regarded as the last statement of both the inner and outer DO-loop. Two CONTINUE statements with different statement numbers will denote the end of the inner and the end of the outer DO-loop respectively. Control transfer from the range of the inner DO-loop to the second CONTINUE statement means the termination of an outer DO-loop iteration. A CONTINUE statement must be used where the last action statement is a control statement. The CONTINUE statement may also be used outside DO-loops. (See 4.)

2.13. GO TO

2.13.1. The Unconditional GO TO is an unconditional branch to a statement label. The form of the statement is:

GO TO label

The Computed GO TO is a selective branch to one of several possible statement labels. The form of the statement is:

GO TO ($label_1$, $label_2$, $label_3$ ... $label_4$), integer variable

When using the computed GO TO, the value contained in the integer variable should be a positive, non-zero number within the range of 1 and the number of places to go in the GO TO list. If the number is 1, then GO TO label 1. If the number is 2, then GO TO label 2 etc. If the number is outside the range, then control goes to EXEC for an error termination. Therefore, the programer may test the variable beforehand and take specific action if desired.

Example:
```
N = I+1
IF (N.LT.1.OR.N.GT.4)  GO TO 100
GO TO (10,20,10,40), N
or  GO TO (M1,M2,M1,M4), N
```

Note: The labels used in the computed GO TO may be switch variables containing statement labels. A variable which has been given the address of a statement number before its reference in a GO TO statement is referred to as a switch variable. See 2.13.2.

2.13.2. ASSIGN STATEMENT

The ASSIGN statement is used only for the assignment of a statement number to a variable which will later be used by a control statement that is not a DO.

The form of the statement is:

```
ASSIGN n TO i
```

where n is a statement number and i is a nonsubscripted integer variable that appears in a control statement.

The execution of an ASSIGN statement presets to "n," the destination of all Control Statements pertaining to "i."

For example:

```
5  ASSIGN  10 to J
   .
   .
   .
6  GO TO J,(1Ø,2Ø,3Ø)
   .
   .
   .
7  GO TO J,(15,25,1Ø)
   .
   .
   .
8  GO TO J
```

Since statement 5 assigns the value 10 to J, statements 6, 7, and 8 are executed as if they had been written:

```
6  GO TO 10
   .
   .
   .
7  GO TO 10
   .
   .
   .
8  GO TO 10
```

Note that

```
ASSIGN 10 to J
```

does not have the same meaning as

```
J = 10
```

In particular, the sequence

```
ASSIGN 10 to J
K = J+1
```

will not produce a meaningful result.

2.13.3. ASSIGNED GO TO

The assigned GO TO is the statement which is used after the ASSIGN statement.

The general form is

GO TO i, (transfer list)

where i is the variable which has previously been set by an assign statement, and the transfer list contains all the possibilities of i separated by commas. The list may be omitted, and the form may be

GO TO i

2.14. MATHEMATICAL FUNCTIONS

FORTRAN provides for the use of certain common mathematical functions, such as square root, logarithm, exponential, sine, cosine, arctangent, and absolute value. In order to make use of a mathematical function, it is necessary only to write its name and follow it with an expression enclosed in parentheses. This directs FORTRAN to compute the named function of the value represented by the expression in parentheses. A complete list of these functions is supplied at a later point in this manual under the discussion of Functions.

A partial list of the mathematical functions follows:

Name	Mathematical Function
EXP	Exponential
ALOG	Natural logarithm
ALOG10	Common logarithm
SIN	Sine of an angle in radians
COS	Cosine of an angle in radians
TANH	Hyperbolic tangent
SQRT	Square root
ATAN	Arctangent; angle computed in radians
ABS	Absolute value

As an example, suppose it is necessary to compute the cosine of an angle named X (a real variable) and to store the cosine in a real variable named Y. We would write:

```
Y = COS (X)
```

The following example illustrates the use of an arithmetic expression as the argument of the function:

```
Y = SQRT(B**2-4.0*A*C)
```

2.15. STOP

The STOP statement has the form:

STOP

or

STOP n

The execution of a STOP statement will terminate execution of the object program and transfer control to the executive system. "n" represents from 1 to 6 alphanumeric characters. If the STOP statement is encountered, the entire STOP statement will be displayed at the console. If a FORTRAN program is normally terminated at a point where a STOP statement is omitted, STOP ENDPRG will be printed at the console and control is transferred to the executive system.

2.16. END

The END statement has the form

END

It is the last physical statement of a program unit. Its appearance tells the compiler to compile the preceding statements as one program unit.

At execution of an END statement, the object program is terminated normally -- indicated by "ENDPRG" on the console. If an END statement is preceded by a statement number, it may be referenced by a control statement.

2.17. EXERCISES

2.17.1. For the following, which are acceptable variable names? Indicate the mode. For those which are not acceptable, indicate why:

XSUB2

JOAN

KNT

GEORGIA

B58

OUT2

MASS.

FLA/7

ABE

2.17.1. (Continued)

Answers

XSUB2	Acceptable	Real	
JOAN	Acceptable	Integer	
KNT	Acceptable	Integer	
GEORGIA	Unacceptable	Real	Too many characters
B58	Acceptable	Real	
OUT2	Acceptable	Real	
MASS.	Unacceptable	Integer	Special symbols not allowed in name.
FLA/7	Unacceptable	Real	Special symbols not allowed in name.
ABE	Acceptable	Real	

2.17.2. Solve the following:

Where A = 2, B = 1, C = 4, D = 2

a. X = A+B-C

b. X = A*B*C**2

c. X = (C/A*D)**2

d. X = (A+B)+(C+D)

e. X = A**3+D

f. X = D*(A+B)

g. X = (B*C+A+B)**D

h. X = A/B/(C*D**A)

2.17.2. (Continued)

Answers

a. X = 2+1-4 X = -1

b. X = 2x1x4**2 X = 32

c. X = (4/2x2)**2 X = 16

d. X = (2+1)+(4+2) X = 9

e. X = 2**3+2 X = 10

f. X = 2x(2+1) X = 6

g. X = (1x4+2+1)**2 X = 49

h. X = 2/1/(4x2**2) $X = \frac{2}{16} = \frac{1}{8} = .125$

2.17.3. Each of the following arithmetic statements contain errors as stated:

1. Y = 2.X+A — The * is missing.

2. 3.14=X-A — The left side must be a variable name.

3. A=((X+Y)A**2 — Not the same number of right and left parentheses. Also, the * is missing.

4. X=1,624,009. *DELTA — Commas are not permitted in constants.

5. -J=1**2. — Integer quantities may not be raised to real powers. Also, variable on left must not be written with a sign.

6. BX6=1./-2.*A**6 — Two operation symbols side-by-side not permitted, even though the minus sign here is not intended to indicate subtraction.

7. Z*X+B=Q — Left side must be a single variable. Should be Q=Z*X+B

8. FNC=CUBRT(X+Y) — No such function supplied. Write FNC=(X+Y)**0.33333333

9. .SQRT(Z)=Z**0.5 — A function name cannot be used as a variable name. Left side must be a variable name.

2.17.4. Each of the following arithmetic statements contains at least one error. State what is wrong with each:

1. -V = A+B

2. 4 = I

3. V-3.96 = X**1.67

4. A*X**2+B*X+C

5. Z2 = A*-B+C**4

6. X = Y+2.0 = Z+9.0

7. R = 16.9X+AB

2.17.5. WRITE ARITHMETIC EXPRESSIONS TO ACCOMPLISH THE FOLLOWING:

1. Add 2 to the current value of the variable named BETA. Make the sum the new value of a variable named DELTA.

2. Subtract the value of a variable named B from the value of a variable named A, square the difference, and assign it as the new value of W.

3. Square A, add the square of B, and make the new value of C the resulting sum.

4. A variable named R is to have its present value replaced by itself raised to the power of 2.

5. Add the values of F and G, divide by the sum of the values of R and S, and square the quotient. Assign this result to P.

6. Increase the present value of G by 2 and replace the present value of G with the sum.

7. Multiply the present value of A by -1.0 and replace the present value of A with the product.

8. Assign to the variable named D a value 1.1 times as great as the present value of the variable named D.

2.17.5. (Continued)

Answers:

1. DELTA = BETA+2

2. W = (A-B)**2

3. C = A*A+B*B
   or
   C = A**2+B**2

4. R = R**2

5. P = ((F+G)/(R+S))**2

6. G = G+2

7. A = -1.0*A

8. D = D*1.1

2.17.6. EXERCISES

1. Write the following numbers as FORTRAN real constants:

   256

   2.56

   -43,000

   10^{12}

   0.000000492

   -10

   -10^{-16}

2. Using the name rule, which of the following names are acceptable for integer variables, and which are acceptable as real variables?

   Write I for integer, R for real, U for unacceptable:

G	_____	BTO7TH	_____
GAMMA	_____	ZSQUARED	_____
GAMMA421	_____	ZCUBED	_____
I	_____	12AT7	_____
IJK	_____	2N173	_____
IJK*	_____	CDC160	_____
J79-12	_____	DELTA	_____
R(2)19	_____	EPSILON	_____
A1.4	_____	A1P4	_____
FORTRAN	_____	ALGOL	_____

2.17.7. EXERCISES

Use the Logical IF and then use the Arithmetic IF to solve both problems below.

1. If four variables (A,B,C, and D) are each equal to 25.0, execute statement number 15. If one or more is not equal to 25.0 execute statement number 30.

2. Assume that values have been computed for three variables X,Y, and Z (no two values alike).

   Write a sequence of statements to make the decisions below:

   If the value of X is:

   (a) greater than Y and greater than Z, go to statement number 10,

   (b) less than Y but greater than Z, go to statement number 20,

   (c) greater than Y but less than Z, go to statement number 30,

   (d) less than Y and less than Z, go to statement number 40.

2.17.7. SUGGESTED SOLUTIONS TO EXERCISES

Logical IF solutions:

```
1.  IF (A.NE. 25.0) GO TO 30
    IF (B.NE.25.0) GO TO 30
    IF (C.NE.25.0) GO TO 30
    IF (D.NE.25.0) GO TO 30
    GO TO 15

2.  IF (X.GT.Y) GO TO 5
    IF (X.GT.Z) GO TO 20
    GO TO 40
  5 IF (X.GT.Z) GO TO 10
    GO TO 30
```

Arithmetic IF solutions:

```
1.  IF (A-25.0) 30,  ,30
    IF (B-25.0) 30,  ,30
    IF (C-25.0) 30,  ,30
    IF (D-25.0) 30,15,30

2.  IF (X-Y)   ,50,5
    IF (X-Z) 40 ,50,20
  5 IF (X-Z) 30,50,10
 50 CONTINUE
```

3. SUBSCRIPTED VARIABLES

3.1. SUBSCRIPTS

It is often desirable to perform computations on groups of numbers or to rearrange or edit them. These groups are called arrays. All of the items within a group are identified by the same symbolic name, and each element within an array may be referenced by writing a subscript (or subscripts) in parentheses after the name of the array in which the element is found. (Refer to 3.4.)

Consider five numbers in a group called "VALUE".

Each number in the array can be referenced by its particular subscript:

Position Number	Values	Subscript Notation
1	1.0	VALUE (1)
2	5.9	VALUE (2)
3	3.6	VALUE (3)
4	2.17	VALUE (4)
5	15.0	VALUE (5)

3.1.1. ALLOWABLE FORMS OF SUBSCRIPTS

Subscripts may be integer constants or integer variables. Three other forms of subscript are permitted. If I stands for a non-subscripted integer variable and L and L' are integer constants, all the allowable subscript forms are as follows:

FORM	EXAMPLE
I	J12
L	3
I±L	K+29
L*I	2*LIMIT
L*I±L'	3*LAST-7

3.1.2. RULES FOR SUBSCRIPT FORMATION

1. The value of a subscript expression is determined each time the subscripted variable is used in the program. It is recommended that this value should not be less than 1, nor greater than the maximum specified in the dimensions of the array, to prevent undesirable effects.

2. A subscript must not be subscripted.

3. A subscript must not contain a function reference.

4. When an array name occurs without any subscripts, all subscripts are assumed to be one. If any one subscript appears, all must appear.

5. Up to seven levels of subscript are permitted.

6. If a variable has more than one subscript, then the total number of variables appearing in the subscripts must be less than 30.

7. Subscripts are separated by commas, enclosed in parentheses, and are appended to the right of the array name to which they apply; e.g., A(3*I-J+1,K).

8. Subscripts are treated as modulo 2^{16}. However, Census conventions allow modulo 2^{15} (32,768) memory cells for both instruction and data in one program.

9. When a subscript is derived as a product of variables, no more than one variable may be an index of a DO. See Chapter 4.

10. The information to the FORTRAN compiler regarding the number and size of subscripts an array contains is supplied by the DIMENSION, COMMON, or type statements. Every subscripted variable must be mentioned in a dimensioning statement before its occurrence in an executable function. See 3.4.

3.1.3. VALID SUBSCRIPTS USED WITH ARRAY ELEMENTS

ARAY (3)

A (I)

B (I+3)

C (JOB-5) provided that 5 < JOB < UPPER INDEX-JOB-5

MATRX (7*IJK) where IJK is the name of a SINGLE integer variable

BUF (3*I+4)

IN (6*M-1)

TABL (I,J*K+3,L-M,6*N) if J or K not a DO index

TAB (M1-M2-M3,LOOK*4,7-J)

N (1,2)

3.1.4. INVALID SUBSCRIPTS USED WITH ARRAY ELEMENTS

ARR(-5) negative not usually permitted

LOT(0) zero not permitted

NEVER(17.23) real not permitted

K3(K(3)) a subscript must not be subscripted

BIG(75000) must not exceed dimensions

NOP(J*K) invalid only if both J and K are DO indexes

HELP(4,2) valid if the array has exactly two dimensions

3.2. ARRAYS

The elements of an array are stored in successive storage units, starting with the first element of the first dimension. The total number of units required for an array is found by multiplying the dimension values specified in a declarator. Thus, the declarator A(3,4) calls for 12 storage units. If double precision is specified, then the number of units for each array is automatically doubled.

The general expression for determining the location of an array element is shown below.

Letters a,b, and c represent subscript expressions in the references; letters A,B, and C represent the dimensions of an array, and m is a multiplier specifying the number of storage units required by each element of the array. Double precision would require two units for m.

DIMENSIONS	SUBSCRIPTS	FORMULA
1	a	(a)m
2	a,b	(a+A*(b-1))m
3	a,b,c	(a+A*(b-1)+A*B*(c-1))m

The following example assumes an array with three dimensions (3,4,2) or (A,B,C) and subscripts given are (2,3,1) or (a,b,c).

The formula is:

```
                (a+A*(b-1)+A*B*(c-1))m
substituting    (2+3(3-1)+3*4*(1-1))1

                2+3(2) + 12(0)
evaluated as    2+6 = 8
```

When the computed subscript is added to the beginning location of an array, the compiler must subtract one. In other words, the eighth cell of an array would be array name plus 7.

3.2.1. ONE-DIMENSIONAL ARRAY = "LIST"

A (N)

List Name No. of elements in List

FORTRAN: A(1) A(2) A(3)...A(I)... A(N)

MEMORY: A A+1 A+2 ...A+I-1...A+N-1

So A(I) = A+I-1

3.2.2. TWO-DIMENSIONAL ARRAY = "TABLE"

B (5,4)

No.of COL No. of ROW

b_{11} b_{21} b_{31} b_{41} b_{51}

b_{12} b_{22} b_{32} b_{42} b_{52}

b_{13} b_{23} b_{33} b_{43} b_{53}

b_{14} b_{24} b_{34} b_{44} b_{54}

FORTRAN: B(1,1) B(2,1) B(3,1) B(4,1) B(5,1) B(1,2) B(2,2) B(3,2)

MEMORY: B B+1 B+2 B+3 B+4 B+5 B+6 B+7

FORTRAN: B(4,2) B(5,2)

MEMORY: B+8 B+9

FORTRAN: B(1,3) B(2,3) B(3,3) B(4,3) B(5,3) B(1,4) B(2,4)

MEMORY: B+10 B+11 B+12 B+13 B+14 B+15 B+16

FORTRAN: B(3,4) B(4,4) B(5,4)

MEMORY: B+17 B+18 B+19

IF B is dimensioned as B(M,N), then the location of

B(I,J) = B+(I-1)+(J-1)*M

3.2.2. TWO-DIMENSIONAL ARRAY = "TABLE" (Continued)

An array may be thought of as one long string broken into smaller sections with individual elements. How these elements are referenced is up to the programer. Consider an array with two dimensions:

DIMENSION A(5,6)

The matrix A will contain six groups with five elements in each group. The first dimension pertains to the elements.

The programer may have a mental picture of the 30 words in memory as five columns and six rows as in the following sequence:

Group 1	1	2	3	4	5
Group 2	6	7	8	9	10
Group 3	11	12	13	14	15
Group 4	16	17	18	19	20
Group 5	21	22	23	24	25
Group 6	26	27	28	29	30

The programer may prefer to think of these words as arrayed in the following order as five rows and six columns.

1	6	11	16	21	26
2	7	12	17	22	27
3	8	13	18	23	28
4	9	14	19	24	29
5	10	15	20	25	30

In any event, no matter what the mental picture, the DIMENSION A(5,6) means that the first subscript should refer to the five cells within a group, and the second subscript should refer to the group number.

A word of caution! When reading records prepared previously, be consistent with dimensions and the mental concept of columns and rows or rows and columns. DIMENSION A(6,5) means five groups of six elements each. DIMENSION A(5,6) means six groups of five elements each.

3.2.3. THREE-DIMENSIONAL ARRAY - SET OF TABLES

C(4, 3, 2)

no. of COL, no. of ROW, no. of TAB

C_{111} C_{211} C_{311} C_{411}
C_{121} C_{221} C_{321} C_{421}
C_{131} C_{231} C_{331} C_{431}

C_{112} C_{212} C_{312} C_{412}
C_{122} C_{222} C_{322} C_{422}
C_{132} C_{232} C_{332} C_{432}

FORTRAN:	C(1,1,1)	C(2,1,1)	C(3,1,1)	C(4,1,1)
MEMORY:	C	C+1	C+2	C+3
FORTRAN:	C(1,2,1)	C(2,2,1)	C(3,2,1)	C(4,2,1)
MEMORY:	C+4	C+5	C+6	C+7
FORTRAN:	C(1,3,1)	C(2,3,1)	C(3,3,1)	C(4,3,1)
MEMORY:	C+8	C+9	C+10	C+11

FORTRAN:	C(1,1,2)	C(2,1,2)	C(3,1,2)	C(4,1,2)
MEMORY:	C+12	C+13	C+14	C+15
FORTRAN:	C(1,2,2)	C(2,2,2)	C(3,2,2)	C(4,2,2)
MEMORY:	C+16	C+17	C+18	C+19
FORTRAN:	C(1,3,2)	C(2,3,2)	C(3,3,2)	C(4,3,2)
MEMORY:	C+20	C+21	C+22	C+23

IF DIMENSION C(N1,N2,N3), then the location of

C(I,J,K) = C + (I-1) + (J-1)*N1+(K-1)*N1*N2

3.3. TABLES AND FILES

3.3.1. A TABLE or FILE or MATRIX may be placed into a two-dimensional array and a SET OF TABLES or SET OF FILES into a three-dimensional array. When setting up such arrays, the following convention should be used:

TABLE

A (No. of COL, No. of ROW)

FILE

A (No. of Words PER RECORD, No. of RECORDS)

SET OF TABLES

B (No. of COL, No. of ROW, No. of TABLES)

SET OF FILES

B (No. of Words PER RECORD, No. of Records PER FILE, No. of FILES)

If the convention is followed, each table will be stored in sequence -- row by row, and each file -- record by record; likewise, each set of tables will be in sequence table by table and row by row within each table, and each set of files in sequence file by file and record by record within each file. This will allow statements like "READ (5,10) A" and "WRITE (6,20) B" to read or display entire arrays in the correct order. Moreover, it will make it possible to operate on individual records, or groups of records within files, or on files or groups of files within sets of files, particularly when using functions and subroutines.

3.3.2. EXAMPLES

1. Suppose we have a file punched on cards in the format shown below, and we want to read this file into memory.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
card no.		n	a	m	e		sex		h	t			w	t		age		)	
1	A	L	L	E	N		M		6	8		1	5	5		3	6	)	6
2	B	E	T	T	Y		F		6	4		1	1	5		3	3	)	1
3	C	I	N	D	Y		F		5	9			9	5		1	2	)	Number of records
4	D	A	V	I	D		M		5	7			9	0		1	0	)	
5	E	L	L	E	N		F		5	3			7	8			8	)	
6	F	R	A	N	K		M		4	6			6	0			4	)	
	Number of words per record = 5																		

The statements below including the formatted READ statement (refer to Chapter 5)

```
      INTEGER  A(5,6)
1     FORMAT (A6,A1,I3,I4,I3)
      READ (5,1)A
```

will read the file into A as shown below:

LOCATION		CONTENTS	
Relative Location	FORTRAN Tag	OCTAL	Interpreted
A	A(1,1)	06 21 21 12 23 05	ALLEN
A+1	A(2,1)	22 05 05 05 05 05	M
A+2	A(3,1)	00 00 00 00 01 04	68
A+3	A(4,1)	00 00 00 00 02 33	155
A+4	A(5,1)	00 00 00 00 00 44	36
A+5	A(1,2)	07 12 31 31 36 05	BETTY
A+6	A(2,2)	13 05 05 05 05 05	F
A+7	A(3,2)	00 00 00 00 01 00	64
A+8	A(4,2)	00 00 00 00 01 63	115
A+9	A(5,2)	00 00 00 00 00 41	33
A+10	A(1,3)	10 16 23 11 36 05	CINDY

3.3.2. EXAMPLES (Continued)

2. Suppose we have a file in an array which we have dimensioned as F(20,500) (five hundred records and twenty words per record) and that we wish to use the sort routine (shown below) to sort records 51 through 75 of the file.*

The statement CALL SORT (F(1,51),25,20,K) will accomplish this, whereas if the convention had not been followed in the main program or subroutine, it would have been almost impossible to do.

```
      SUBROUTINE SORT (A,N,W,K)
C     A=ARRAY TO BE SORTED
C     N=NO OF RECORDS
C     W=NO OF WORDS PER RECORD
      ...
      DIMENSION  A(W,N)
      ...
```

3.4. DIMENSION LIMITS

3.4.1. Arrays may have 1-7 dimensions. Array sizes may be specified in DIMENSION, COMMON or type statements. Dimension limits may be unsigned integer constants or variables previously defined in PARAMETER statements. Refer to Chapter 7, "SPECIFICATION STATEMENTS."

EXAMPLES:

```
INTEGER A,B,C                    |  PARAMETER S = 100
DIMENSION A(2),B(100),C(5)       |  DIMENSION X(S),Y(S,2)
INTEGER A(2),B(100),C(5)         |  DIMENSION  AR(5,4,8,6)
INTEGER T                        |  PARAMETER N = 50
COMMON/ALPHA/M(20),T(10,6)       |  REAL  K(N,N),P(30),L(N)
INTEGER A(100),B(3,3)            |  LOGICAL  R,S,T
REAL  I(2,5),J(9),K(30)          |  DIMENSION R(10),S(3),T(2,2)
COMPLEX  X(50),Y(4,4,4)          |  COMMON /MMM/R,S,T
```

* A sort will arrange or sequence items according to a specified set of rules.

3.4.2. It is permissible to refer to array names in I/O statements or as arguments of functions and subroutines. Otherwise, any reference to an array should contain the same number of subscripts with which it was dimensioned.

EXAMPLES:

```
INTEGER  A(100)                    REAL  M(10,10)
   ...                                ...
READ(1) A                          CALL ZERO  (M,100)

INTEGER A(100)                     DIMENSION N(50),P(60)
   ...                                ...
READ(1) (A(I),I=1,100)             WRITE (5,6) A,N,P,X,Y
```

Illegal Operations:

```
DIMENSION X(5,3)                   INTEGER  Q(500)

X(10) = 0                          Q = Q+1 (if Q(2) is intended)
```

3.4.3. It is legal to refer to an array with dimensions exceeding those with which it was dimensioned, making possible zero and negative subscripts. This procedure is not recommended.

EXAMPLE: COMMON A(5),B(3),C(4)

```
A(1) A(2) A(3) A(4) A(5) B(1) B(2) B(3) C(1) C(2) C(3) C(4)
 "    "    "    "    "                  "    "    "    "
B(-4) B(-3) B(-2) B(-1) B(0)            B(4) B(5) B(6) B(7)
```

...

```
      DO  10  I=1, 12
10    A(I) = 0
```

or

```
      DO  10  I= -4,7

10    B(I) = 0
```

3.4.4. At the time of computation, it is possible to assign data into variables or into named arrays. This is done by means of the DATA statement. This subject is discussed in further detail in Chapter 7 "SPECIFICATION STATEMENTS".

3.5. MATHEMATICAL NOTATION

Symbols seen frequently in statistical problems are:

$\sum$ - a sum of numbers is desired

$\prod$ - a product of numbers is desired

! - the factorial is desired, i.e. if a number is given as 5! the product of $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ is the factorial.

These symbols may be written with an index with ranges as in the example below:

$$\sum_{i=1}^{n} X_i$$

which can be interpreted as "add all those values in the array X whose subscripts are between 1 and n inclusive." If the subscript i were set to some other value, then the range would begin wherever specified.

Other examples are:

a) Derive the sum of the squares of n values in array X. beginning with X(1) and including X(n).

$$\sum_{i=1}^{n} X_i^2 = X(1)^2 + X(2)^2 + X(3)^2 \ldots\ldots + X(n)^2$$

b) Derive the sum of the products of n values in the arrays X and Y.

$$\sum_{i=1}^{n} X_i Y_i$$

c) Square the sum of n values in array Y.

$$\left(\sum_{i=1}^{n} Y_i \right)^2$$

For an understanding of how these symbols may be interpreted in FORTRAN programs read chapter 4 (The DO Statement) and see the example in 5.8.12.

4. THE DO STATEMENT

4.1. DO

The DO statement allows the programer to construct a program loop. In other words, it causes the subsequent line(s) of the program to be repeated a specified number of times -- incrementing or decrementing a certain value for each execution.

The first statement in the range of a DO must be one that can be executed. It is permissible for the range of one DO (the outer DO) to contain another DO (the inner DO).

The processor is provided with the following information for each DO:

a) the label of the last statement in the group to be repeated

b) a control variable representing the loop counter and assigned the initial value

c) the initial value for the loop counter (par 1)

d) an upper limit or maximum value which is not to be exceeded by the loop counter (par 2)

e) an increment value by which the loop counter is to be modified after each execution (par 3) -- this value must not be a zero.

The general form of the DO statement is as follows:

```
                    par1,par2,par3
DO label V =         M,   N,   I
```

where label is the label reference of the last statement in the loop, V is the variable of the loop, M and N are the minimum and maximum value, respectively, taken on by V, and I is the increment index. If I is omitted, the increment is assumed to be 1 or -1 if par1 > par2 and both are integer constants.
For example,

DO	50 label	I Control variable	=	1, Initial value	10 Upper limit Value	,1 Increment value

loops ten times from this statement to statement number 50, incrementing I, by one each time.

4.1. DO (Continued)

The parameter values may be integer constants, a parameter variable or nonsubscripted integer variables. If integer variables are used, they must be defined prior to the execution of the DO statement. The control variable and the parameter values must not be modified by the programer during the execution of the DO-loop; however, the value of the control variable (or index variable) may be used advantageously.

Correct Example:

```
    DO    50   I = 1,N
    }
    }
    }
50 terminal statement
```

Incorrect Form:

```
    DO   50   I = 1,N(J)
```

Subscripting is not allowed

Incorrect Form:

```
DO      50    I= 1,N+1
```

Expressions are not allowed. The value must be derived before its use.

4.2. DO-LOOPS

The set of statements which follows the DO statement including the statement with the referenced label is called the range. The DO statements plus the range is the DO loop. A DO loop will be executed at least once, regardless of the index. The value of the index is undefined if the DO loop terminates because the index is exhausted. Throughout the range of the DO loop, the index variable is available both in subscripts and as an ordinary integer variable. The programmer, however, should not attempt to change the value of the index variable.

If the DO loop index is exhausted, the index will no longer be available for use.

For example:

```
          DO 1Ø  J = 1,5,3,
           .
           .
           .
      10
           .
           .
           .
        L = P+J
```

4.2. DO-LOOPS (Continued)

the last value set into J has been lost and must be set to some value again before it may be referenced.

To permit the DO statement to maintain control of the loop, the terminal statement must logically be executable, and the terminal statement must not be any of the following control statements:

a) A GO TO in any form

b) An arithmetic IF

c) A DO

d) A STOP

e) A RETURN

However, if it is necessary to use one of the above statements as the last statement in a DO loop, it may be done by following the statement with a dummy statement called CONTINUE. The form is: CONTINUE

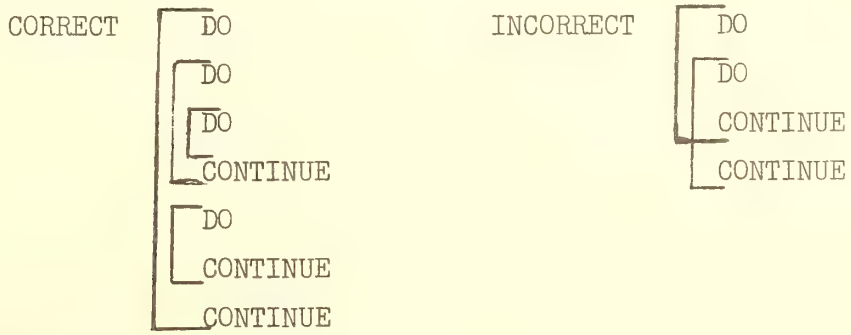
The CONTINUE statement is then the terminal statement and hence must be labeled as the last statement in the range.

4.3. NESTED DO-LOOPS

Among the statements which may occur within the range of a DO-loop is another DO statement. When this is so, the following rules must be observed:

1. If the range of a DO-loop contains another DO-loop, the range of the second must be entirely within the range of the first.

2. The range of the inner DO-loop may, however, contain the last statement in the range of the next outer DO-loop. Such a set of DO-loops is called a nest.



4.3. NESTED DO-LOOPS (Continued)

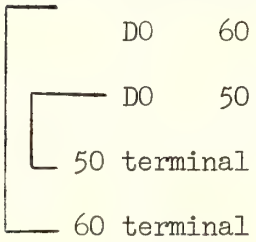
The following rule applies to transfers into the range of a DO-loop:

Rule: No transfer is permitted into the range of a DO-loop from outside its range.

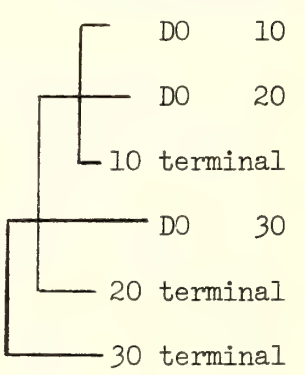
Exceptions:

If control has been transferred from the innermost DO-range of a nest of DO-loops to a section of the program which is not in the range of another DO-loop, and which does not change any of the indices or indexing parameter of the nest, then control may be transferred back to the range of the innermost DO-loop from which control was originally transferred. A transfer from an inner DO to an outer DO is allowed.

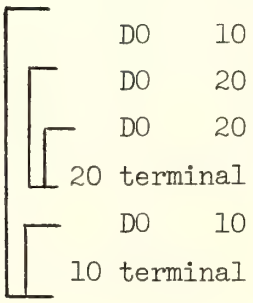
Correct Use



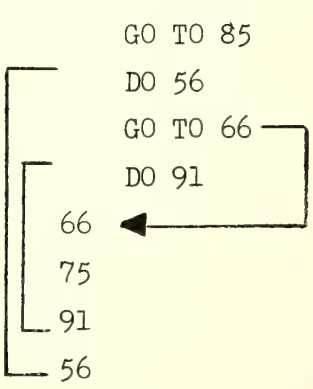
Incorrect Use



Correct Use



Incorrect Use



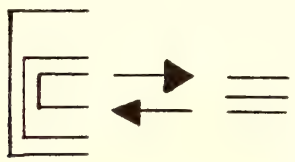
4.3. NESTED DO-LOOPS (Continued)

Correct Use of Extended DO-loop:

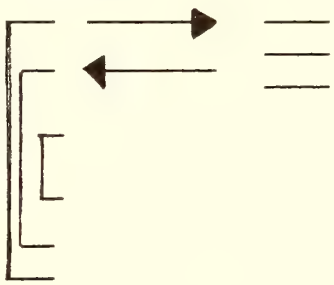
```
     DO  50  I = 1,10,2
     .
     .
     .
     IF (expression) GO to 11 ⟶  11  A = B*C

     A = B*C+3.14                        GO TO 78

 78  X = SIN(A)
     .
     .
     .

 50  CONTINUE
```

Note: An extended DO-loop can be executed only from within an innermost nest. The correct and incorrect execution of an extended DO is shown below

Correct



Incorrect



Note: Reference to a function or a subroutine within a DO-loop is permitted from any point within the loop.

4.3. NESTED DO-LOOPS (Continued)

When an array containing more than one dimension is referenced within nested DO-loops using the DO-indexes as subscripts, then care must be exercised to achieve the desired results.

One complete cycle of the innermost loop is executed for every execution of the next outer loop. For example, it is desired to create row-totals and column-totals for a matrix N with six (6) rows and 10 columns per row. At the Bureau of the Census totals are usually displayed in the first row and first column of printed tables. Various methods are used for developing totals. Some programers merely leave enough room in each table matrix to allow for row and column totals. A more sophisticated method is to develop the individual tally cells required for each table and to set aside one area wherin totals may be created for one table at a time and reused as each succeeding table is prepared for output. In any event, areas set aside for totals should be filled with zeros before any summing occurs. In the example below, row and column totals are a part of the matrix.

```
      DIMENSION N(10,6)
      .
      .
      .
      DO   100   J=2,6
      DO   100   I=2,10
100   N(1,J)=    N(1,J)+N(I,J)
      DO   200   I=1,10
      DO   200   J=2,6
200   N(I,6)  =  N(I,6) + N(I,J)
```

This sequence of this coding is as follows:

1. Using J as a subscript for the row, and I as a subscript for the column, add columns 2 through 10 to column 1. Then increment J to the next row.

2. Repeat step 1 until cross-totals for rows 2 through 6 have been created.

3. Using I as a subscript for the column, and J as a subscript for the row, add rows 2 through 6 to row 1. Then increment I to the next column.

4. Repeat step 3 until all 10 columns have been summed.

The final result would be to create column and row totals in a matrix in subscripted locations as shown:

TOTALS

TOTALS	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1
	1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2	10,2
	1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3
	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4	10,4
	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5
	1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6	9,6	10,6

4.4 MATERIALIZATION OF THE INDEX VARIABLE

If the index of the DO is used within the range of a DO-loop (except as a subscript) then the storage location associated with the index variable will contain the current value of the index variable throughout the range of the DO-loop.

The following uses of the index variable will cause materialization to occur. New terms below will be explained in chapters 6 and 7.

1. The occurrence of the index variable other than as part of a subscript.

2. A CALL statement to an external subroutine and the index variable is in COMMON.

3. A control statement referencing a statement outside of the range of the DO-loop.

4. A reference to an internal subprogram and the index variable is global.

5. A reference to an ABNORMAL function, and the index variable is in COMMON.

6. A reference to the statement function and the index variable is in COMMON.

4.5 Frequent Uses of a DO-LOOP

One of the most frequent uses of a DO-LOOP is to reference various elements of an array. As the index of a DO changes, the value of the subscripts may also be changed. For example, if an operation is to be performed on the first three elements of an array, the index may proceed from 1 to 3 to pick up those elements, but not the others.

```
      DO  10     I = 1,3
10    A(I)         = 2*A(I)
```

The first three elements would be doubled, while the remaining ones are unchanged. The loop is equivalent to the execution of the statements

```
A(1) = 2*A(1)

A(2) = 2*A(2)

A(3) = 2*A(3)
```

The same results could be obtained by starting with the third element and working in descending order.

```
      DO  1Ø I = 3,1
1Ø    A(I) = 2*A(I)
```

This is equivalent to the following statements:

```
A(3) = 2*A(3)

A(2) = 2*A(2)

A(1) = 2*A(1)
```

If a jump instruction transfers control outside the range of a DO-loop, then the storage location associated with the index variable will contain the value of the index variable at the time the transfer occurred.

```
      DO      10     I=1,5
       K = I+1
      IF (K.EQ.2) GO TO 15
      ____
      ____
      ____
10    CONTINUE
      ____
      ____
      ____
      ____
15    J = J*I
```

The variable "I" will contain the value 1 when control is transferred on the first execution of the DO-loop.

4.6. Exercises Using The DO INSTRUCTION

1. Write instructions to move contents from locations 1-20 in TABCO to the corresponding locations in TABST.

2. Write a DO-loop to make a count of all the numbers greater than 10.0 in a 1000-number array called A.

3. Use a single DO-loop (a single index) to clear an array called RUST (100 locations) and an array called MARK (100 locations) to binary zeros.

4. Write a program to set each value of an array called INDEX equal to the order of that value in the array (that is, INDEX (1) = 1, INDEX (2) = 2, etc.). The INDEX has a total length of 2000 numbers.

5. An array called BOOZE contains 500 numbers. Move the contents of locations 1-250 into locations 1-250 in an array called GIN and the remaining 250 numbers into positions 1-250 of an array called VODKA.

6. An array called ROX has 200 numbers. Write instructions to determine number of positive numbers, number of negative numbers, number of zero numbers.

Exercises Using the DO INSTRUCTION (cont.)

7. An array called FAN contains 100 numbers. Using a built-in function, compute the square root of each number, and store it in the corresponding location of an array called RUTE.

8. Twenty (20) numbers are stored in an array called ORDER. Reverse the order of the numbers -- i.e., place them in reverse order in the same array.

9. Write a DO-loop to copy the contents of an array called A into an array called B, both of which are 1000 numbers long, placing the numbers in the reverse order to that which they occupied in the original array. That is, B(1000) will contain A(1), B(999) will contain A(2), etc.

10. An array of numbers called A contains 1000 numbers. Find the largest number and store it at BIGA. Find the element number and store it at BIGB.

11. Two one-dimensional arrays named X and Y contain 50 elements each. A variable named XS is known to be equal to one of the elements in X. When equality is found between XS and an element in X, pick up the corresponding element in Y and place it in the variable YS.

4.6. Suggested Solutions to Exercises

Note: It is assumed that all arrays are dimensioned prior to use and summary cells are cleared to zero.

1.
```
      DO   10   J=1,20
   10 TABST(J) = TABCO(J)
```

2.
```
      DO   20   J = 1,1000
      IF (A(J)-10.0)20,20,
      N = N + 1
   20 CONTINUE
```

3.
```
      DO   30   J = 1,100
      RUST(J) = 0
   30 MARK(J) = 0
```

4.
```
      DO   40   I = 1,2000
   40 INDEX(I) = I
```

5.
```
      DO   50   K = 1,250
      GIN(K) = BOOZE(K)
   50 VODKA(K) = BOOZE(K+250)
```

6.
```
      DO   60   M = 1,200
      IF (ROX(M))    ,20,30
      NEG = NEG+1
      GO TO 60
   20 NZERO = NZERO+1
      GO TO 60
   30 NPOS = NPOS+1
   60 CONTINUE
```

Suggested Solutions to Exercises (cont.)

```
7.        DO   70   N = 1,100
     70   RUTE(N) = SQRT (FAN(N))

8.        DO   80   I = 1,10
               M =  ORDER (21-I)
               ORDER (21-I) = ORDER (I)
     80   ORDER (I) = M

9.        DO   90   L = 1,1000
     90   B(1001-L) = A(L)

10.       BIGA = A(1)
          BIGB = 1
          DO   10   K = 2,1000
          IF (A(K).LT.BIGA) GO TO 10
          BIGB = K
          BIGA = A(K)
     10   CONTINUE

11.       DO   100  J = 1,50
          IF (X(J) - XS)    ,200,
    100   CONTINUE

     c    If the loop terminates without finding equality,
     c    then an error condition exists
     200  YS = Y(J)
```

5. FORTRAN I/O

5.1. FORMATTED READ AND WRITE

5.1.1. FORMATTED READ

A read statement controls the transfer to memory of information from an external source. (At Census, magnetic tapes are used as input to FORTRAN programs.) If the original data is card input, a service routine is used to place the data on tape. This routine is called CENSER, with option CT (Card-to-tape function). If the input data was produced in a previous run by a formatted write on unit -3, then the data may be read on unit 5 with a formatted read. The form of the formatted READ is:

READ(5,label)$name_1$, $name_2$, $name_3$ ....

label refers to the label of a FORMAT statement which informs the program how to pick up fields of information and how to handle each one. Name is a variable, array, or array element name into which the data will be placed. The 5 is fixed as the unit number which is usually used for reading formatted data.

An example of a READ statement is:

```
READ(5,20)INT,XRAY,COUNT,TOP

20  FORMAT (I6,3F12.2)
```

The action taken by this read is to convert the first six characters of input to an internal binary form as an integer number, placing the result in the location INT. The next 12 characters will be converted to a floating point number, taking into account the fact there are two decimal places. This result is placed in the REAL location named XRAY. These steps are then repeated to place results into COUNT and TOP. The 3 causes the floating point format to be used 3 times before going to the next format.

In a formatted READ, the FORMAT specifications are interpreted until the READ list is completely filled with data. If necessary, the FORMAT is rescanned and another record is read with each rescan. See Rescanning, 5.4.4. for the rule to determine the starting point.

5.1.2. NOTE ON METHOD TO DETECT END-OF-DATA ON TAPE

It is often the desire of the programer to do something special after all the data has been read in. If no indicator is checked by the programer, the FORTRAN procedure terminates the program when an end-of-tape condition is detected. Therefore, it has been the practice at Census to create a record of all 9's or other sentinel as the last data record. A check for this special record can be made after each READ. When it is detected, the programer may then perform any action desired before terminating his program.

5.1.3. FORMATTED WRITE

A WRITE statement controls the transfer of information from storage to a peripheral device. (At Census, magnetic tapes are used as output.) If it is desired to print the output, a service routine is used. This routine is called CENSER, and the option to specify is TP (Tape-to-print function, Lion External Format). Unit 6 is used at Census for formatted writes which are to be printed with a maximum of 128 characters per line. Unit -3 is used to produce a tape which may be read in as input to a subsequent program and the output consists of 80 character card images or 14 word records. The form of the FORMATTED WRITE is:

WRITE(Unit, label) $name_1$, $name_2$, $name_3$....

where label refers to the label of a FORMAT statement which informs the program how to handle each field of data it picks up from the variable specified in the name list.

In a formatted write, the FORMAT specifications are interpreted until the WRITE list is completely exhausted of data. If necessary, the FORMAT is recanned and another record is written to correspond with the rescan. See 5.4.4. for the rule to determine the starting point.
EDTCPY (14.8.) should be used to prepare a tape for off-line printing.

5.1.4. CONTROL OF PRINT FILES

The first character of a printer record, produced by the interaction of a WRITE and a FORMAT statement, does <u>not</u> appear on the printed output page; it is used to control line spacing before printing.

FORTRAN provides an interpretation of four carriage control characters:

blank	(1Hb) or (1X)	Print the record on the next consecutive line.
0	(1H0)	Print the record on the second consecutive line.
1	(1H1)	Print the record of the top of the next page.
+	(1H+)	Print the record on the current line; this provides a means of writing two or more records on one line. When using the +, two or more characters should not be printed in the same position. Spaces may be used in those positions where over printing is not desired.

5.1.4. CONTROL OF PRINT FILES (Continued)

If the first character of a print record is not one of the above, the action taken is unspecified and usually erratic.

It is a good practice to specify an H field descriptor at the start of a FORMAT statement for each printer record.

5.1.5. READ AND WRITE NAME LISTS

There is a one-to-one correspondence between the VARIABLE names specified in a basic READ or WRITE statement and the data fields to be transferred into these variables. This correspondence is established by associating a name and a field starting with the leftmost position and progressing to the right.

Unlike a variable name and an array element name, an array name implies all the elements in that array and, therefore, will be associated with more than one field.

EXAMPLE:

```
      DIMENSION GRUP(3),X(20)

      READ(5,98) A,X(4),GRUP

   98 FORMAT (5F6.1)
```

Assume the input has 5 fields of data each containing 6 digits with 1 decimal place. The fields contain 5.6, 78.0, 3.2, 3.4, and 3.5. Then after the execution of the above READ values are in the following:

```
    A =  5.6

 X(4) = 78.0

GRUP(1) = 3.2

GRUP(2) = 3.4

GRUP(3) = 3.5
```

A value may be read into a variable and that value used as a subscript to a name to inform the program where to place data.

Example:

```
                INPUT DATA      ⌈ 5     8.0     15     2.1

                READ(5,72)   I, BLUB(I), C, X(I)

           72   FORMAT (I1, F6.1, I6, F6.1)
```

5.1.5. READ AND WRITE NAME LISTS (Continued)

Name I is being assigned a value of 5; thus the names BLUB(I) and X(I) in the list become BLUB(5) and X(5) respectively. After execution, the following variables will contain values.

```
      I    =   5
BLUB(5)    = 8.0
      C    =  15
   X(5)    = 2.1
```

5.2. IMPLIED DO

A basic READ-or-WRITE statement name list, in addition to specifying names previously discussed, may also include an implied DO list. This is used in operations involving arrays. An array name in a list causes all the elements of the array to be transferred. An implied DO allows selected sections of an array to be transferred. The implied DO notation is similar to the DO statement, but it does not require the word DO and a terminal statement. The form of the implied DO specification is:

$$i = par_1, par_2, par_3$$

where i is the subscript variable

par_1 is the first subscript value

par_2 is the terminal subscript value

par_3 is the increment for the subscript

An implied DO list includes the specification and the names of the arrays (with a subscript(s)) to be operated on. The entire list must be enclosed by parentheses. For example, if 15 elements are to be transferred from a one dimensional 100 element array named SAM, the notation appears as follows:

```
(SAM(I), I = 10,24,1)
```

An example of the notation for transferring 25 elements of a two dimensional array is as follows:

```
((SAM(I,J),I = 1,5,1), J = 1,5,1)
```

In a basic statement such as READ, an implied DO may appear as follows:

```
READ (5,30) AL,INT,(SAM(I),I = 15,24,1), J,K
```

5.2. IMPLIED DO (Continued)

The transfer of information occurs in the following sequence:

1	AL
2	INT
3	SAM(15)
4	SAM(16)
	.
	.
	.
	.
12	SAM(24)
13	J
14	K

General forms of implied DO's are as follows:

One-dimensional form:

$(name_1(i), name_2(i),..., name_n(i), i = par_1, par_2, par_3)$

Two-dimensional form:

$((name_1(i,J), name_2(i,J),...., name_n(i,J), i = par_1, par_2, par_3),$

$J = par'_1, par'_2, par'_3)$

Three-dimensional form:

$(((name_1(i,j,k), name_2(i,j,k),....name_n(i,j,k),$

$i = par_1, par_2, par_3), j = par'_1, par'_2, par'_3),$

$k = par''_1, par''_2, par''_3)$

An example of implied DO notation using one subscript is as follows:

```
DIMENSION D(8)
READ( 5,65) A,B,(D(I),I = 1,4,1)
```

This is the equivalent of writing:

```
READ ( 5,65) A,B,D(1),D(2),D(3),D(4)
```

5.2. IMPLIED DO (Continued)

An example using two subscripted names in an implied DO is as follows:

```
READ( 5, 65)  D(2), (B(I+2),C(I,1),I = 2,6,2)
```

The transfer of information occurs in the following sequence:

1	D(2)
2	B(4)
3	C(2,1)
4	B(6)
5	C(4,1)
6	B(8)
7	C(6,1)

Warning: It is illegal to use a variable in an implied DO range to define a parameter in an implied DO. For instance, the following is an incorrect notation:

```
READ( 5,70)  (A(I),J,I = 1,J)
```

The value associated with J would change every time the loop was executed. This is an attempt to redefine the DO parameter J and is illegal.

5.3. UNIT Assignments for FORTRAN READ/WRITE Statements

Type of Data	Unit Number	Tape Name	Channel	Tape Type
*card image input	5,-1	READT	MTO	IIIA
*card image output	-3	PNCHT	MTO	IIIA
**printer image	6,-2	PRNTT	MTO	IIIA
unformatted	1-4	AA-DA	MAO	IIA
unformatted	7-9	EA-GA	MAO	IIA
unformatted	10-16	AT-GT	MTO	IIIA
unformatted	17-18	AC-BC	MCO	4C

*card images contain 80 accessible characters
**printer images contain 128 accessible characters including the carriage control character

Note: Census programers should restrict themselves to the use of READT, PNCHT, and PRNTT tapes when using FORTRAN READ/WRITE statements because of differences in tape formats.

5.4. FORMAT STATEMENT

The FORMAT statement is used to describe how the information is arranged on input or is to be arranged on output. For each value transmitted there must be a corresponding field specification which lists the kind of information the field contains (in terms of its internal representation), and what it "looks like" externally. A general form of the FORMAT statement follows:

label FORMAT (descriptor$_1$, descriptor$_2$, ..., descriptor$_n$)

where LABEL is the numerical statement label referred to by the READ or the WRITE statement, and the DESCRIPTORS are the field specifications for the individual pieces of data. The first piece of data takes the format of DESCRIPTOR$_1$, the second, DESCRIPTOR$_2$, etc.

For example:

```
10    FORMAT (I6,F6.2)

      READ(5,10)M,XNUM
```

reads in a piece of data as a six-character integer and stores it in M; then it reads the second piece of data as a six-character fixed point number with two decimal places and stores it in XNUM. However, this statement also has additional features which provide conveniences to the programer. The FORMAT statement is referenced by the statement number of either the READ (UNIT, format designation) List,or WRITE (UNIT, format designation*) List statement. The FORMAT statement provides the following descriptive information or specifications:

a) description of the field or fields of data to be transferred

b) convenient notation for describing groups of similar fields

c) convenient notation for controlling the transfer of records of data

*Note: In lieu of a FORMAT statement, the formatting specification may be contained in an array. A pair of left and right parentheses should bound the specifications. The array may be filled by a previous READ before its reference in the object program.

5.4.1. FIELD DESCRIPTORS

GENERAL RULES:

1. A letter (I,F,E,D,G,Q,L,A,H, or X) should be included to designate the kind of information and how it is to be handled. (See 5.4.5.)

2. There should be a number to designate how many characters or card columns are in use. This will be referred to as w.

3. On input a sign, if any, must be the first non-blank character of a field. The use of a plus sign is optional; if no sign appears, the number is considered positive. Blanks are taken to be zeros.

4. On output, the number will appear at the right of the output field if more character positions are specified for the field than there are characters in the number. If too few characters are specified, the output area will be filled with asterisks.

5. In all types, except H and X, it is permissible to specify that the same field specification applies to several successive fields, by writing a repetition number referred to as r in front of the field specification.

6. If the programer would like to change the number of decimal places of E, F, or G formatted data, he may employ a scale factor within his formal statement. The editing codes of the formes E, F, and G may be written in the form pPrEw.d where the Ew.d is the normal field specification, the r is the repetition factor, and the p is the scale factor, a signed integer constant. This form may be used for both input and output.

   In printing, the appearance of a scale factor p has the following effect:

   a. List values associated with the field of the form rFw.d are multiplied by 10p before printing. A negative scale factor will move the decimal point to the left.

      Example:

VALUE	SPECIFICATION	PRINTING
12764.31613	-2PF12.7	127.6431613
	2PF12.3	1276431.613

5.4.1. FIELD DESCRIPTORS (Continued)

6. (continued)

b. A list value associated with a field of the form rEw.d is printed with its mantissa multiplied by 10^p, and its exponent decreased by p. In other words, the field is changed in form, but not in value, on printed output. A scale factor has no effect on input with an E specification.

EXAMPLE:

VALUE	SPECIFICATION	PRINTING
-12764.31613	E13.6	-0.127643E05
	2PE14.6	-12.764316E03
	-3PF7.2	-12.76

7. Whenever a scale factor is written with a D,E, or F field specification, it automatically applies to all succeeding D,E, and F field specifications, until some other scale factor is encountered. If a given scale factor is applied to one field specification only, the next field specification must have a ØP scale factor.

8. In the E,F, and G forms, there should be a number to designate how many decimal places are in use. This will be referred to as d.

9. Data to be treated as octal must be given real or integer variable names. Internally, such data will be presented as a 12 digit octal number.

10. Repetition of Groups of Editing Codes: If it is desired to edit more than one group of elements of a List in precisely the same manner, and the corresponding groups of editing codes would occur in succession in a FORMAT specification, then it is possible to further shorten the FORMAT specification. The group of editing codes to be repeated may be written only once, enclosed in parentheses and prefixed by an unsigned integer constant, n, indicating the number of times repetition of the group is desired.

Example: FORMAT (4(F12.1,E5.2)) is equivalent to:
FORMAT (F12.1,E5.2,F12.1,E5.2,F12.1,E5.2,F12.1,E5.2)

5.4.1. FIELD DESCRIPTORS (Continued)

11. Multiple Record Format Specification

If a group of editing codes is followed by a / (slash), then the remainder of the record being edited is ignored on input or filled with spaces on output, and the editing codes following the slash (if any) are used to edit the next record.

Thus, FORMAT (3F9.2,2F10.4/8E14.5) could edit a two-line block of printing (two records), the first line of which would have its editing controlled by the codes 3F9.2,2F10.4 and the second line of which would have its editing controlled by the code 8E14.5. Entire records may be ignored on input or blank (skipped) records may be produced on output simply by writing consecutive slashes.

For example:

```
4122 FORMAT (8H△SPIN  IS,2E10.5,2A6/6F7.2 /////)
```

used with a WRITE (6,4122) would produce a line controlled by 8H△SPIN IS ,2E10.5,2A6, a line controlled by 6F7.2 and 4 lines of spaces.

Note: If it is desired to skip n lines, then there must be n+1 slashes.

5.4.2. TABLE OF EDITING SYMBOLS

SYMBOL	EDITING ACTION			EDITING CODE
	Internal		External	
I	Integer variable	to/from	decimal integer	Iw,rIw
E	Real variable	to/from	floating point decimal number	Ew.d,rEw.d
D	Double-precision	to/from	floating point decimal number	Dw.d,rDw.d, pPrEw.d
F	Real variable	to/from	fixed point decimal number	Fw.d,rFw.d, pPrFw.d
G	Real variable	to/from	floating point or fixed point decimal number	Gw.d,rGw.d
O	octal-digit	to/from	octal digit	Ow,rOw
L	Binary bit		T/F	Lw,rLw
A	6-bit Character	to/from	6-bit character	Aw,rAw
H	supplied by FORMAT		6-bit character	wH
X	not stored		Skip/Space	wX

Notation:

w is width of field

d is number of decimals

p is amount of scaling by power of 10

P is scaling

r is number of repetitions

. is decimal point to separate width of field and number of decimal places

5.4.3. OBJECT TIME FORMAT STATEMENTS

If the programer does not know the field specifications of the data when he writes the program, it is possible to input these specifications with the other data. To do this, an array to hold the format specifications in the form of alphameric data must be established. At object time, these formats are read into this array, and the formats are then referenced by name in the READ or WRITE statement.

For example, if the format for the output of four elements, I, J, K, L, differs with every set of data, a unique format enclosed in parentheses may be contained on the first data card as follows:

(I10,I16,1PE2Ø.8,0PE16.2)

FMT is dimensioned as a one-dimensional array with five elements, which will be considered as alphameric variables and therefore can hold 3Ø characters (five "words" of six characters each) to hold this information. This variable FMT will then contain the format information and may be referenced like a format label.

```
      DIMENSION  FMT(5)
      READ (5,209)FMT
209   FORMAT (5A6)
      .
      .
      .
      WRITE (6,FMT)I,J,K,L
```

5.4.4. RESCANNING

If the number of items mentioned in a READ statement is larger than the number of specifications in the FORMAT statement, it will return to some portion of the FORMAT statement to find the remaining specifications.

For example:

```
      READ (5,10)A,B,C,D
      FORMAT (I6,F6.1)
```

will associate A with I6, B with F6.1 and then return to the beginning of the FORMAT statement to associate C with I6 and D with F6.1. If more data were to be read, it would continue in this manner until all the variables in the READ statement had been filled.

One use of the rescan is to force the reading of several records to fill an array. When the first scan of the FORMAT is completed, a new record is brought in, and it is rescanned. This continues until the array list is filled with data. For example, it is desired to read six records, pulling off three six-character fields from each one, to fill an array of 18 cells.

```
      DIMENSION A(18)
      .
      .
      .
      READ (5,10)A
10    FORMAT (3F6.1)
```

When a format must be rescanned when using a WRITE, the rescan initiates a new output record. For example, the following WRITE statement would create 20 records:

```
      DIMENSION  N(20)
20    FORMAT     (I6)
      WRITE (6,20)N
```

If it were desired to create two records of 10 fields each, the format should read:

```
20    FORMAT (10I6)
```

It is possible to specify a special format for the first (one or more) records and a different format for all subsequent records. This is done by enclosing the last record specifications in parentheses.

5.4.4. RESCANNING (Continued)

Excluding the rightmost parenthesis of the FORMAT specification, the restart point is the left parenthesis that matches the rightmost parenthesis. If an integer constant for repetition immediately precedes the rescanning entry point, it is also rescanned.

In the following examples, an arrow indicates the rescan point.

```
FORMAT (F5.1,G20.10)
        ↑

FORMAT (F11.1, (I4,E10.2))
                ↑

FORMAT (2 (F11.1), I4,/E10.2)
        ↑

FORMAT (2(E6.1,I4),3(I2,I4,E6.2))
                   ↑

FORMAT (2(E6.1,I4),3(F7.2,E5.1(I3,I2)))
                   ↑

FORMAT ((G5.1,I3),2(G6.2,F5.1,(I3,I2)),I4,P6.1)
                ↑
```

If the end of a variable name list is reached but there remain H, X or/ specifications, they will be processed before the input-output operation is terminated.

If the name list is exhausted before the specifications, the input-output operation is complete.

5.4.5. SPECIFIC INFORMATION RE EDITING CODES

5.4.5.1. Iw

a. "I" means integer data is involved

b. The maximum value which can be read or written is ± 34,359,738,367 (11 decimal digits plus sign)

c. Valid input characters for the field are:

Space or blank and the numerics Ø-9.

A plus (+) or minus (-) sign is also acceptable as the character preceding the first numeric character.

d. On input, the value is converted to pure binary and stored, right justified, in one computer word. If the field width is greater than 11, then the leading positions may only contain a sign or spaces or blanks.

If the entire input field is blank, a value of -0 is stored.

e. On output, the value is converted from pure binary to integer. If the value is negative, a minus sign (-) will appear immediately preceding the most significant digit. If the value is positive, no sign will appear. If the field width is larger than required for the value, the field will be filled out to the left with spaces. If the field width is not large enough to contain the value including (minus) sign, the field will be filled with asterisks to indicate overflow.

The following examples show how each of the quantities on the left is printed according to the format code I3:

INTERNAL VALUE	PRINTED VALUE
721	721
-721	***
-12	-12
568114	***
0	⊬⊬0
-5	⊬-5
9	⊬⊬9

5.4.5.2. Fw.d

a. The "F" phrase specifies that the next w characters are to be interpreted as a fixed-point number. The data may be either a real variable or one part of a complex variable.

b. The w is the total number of characters which are to be interpreted. The total field length reserved must include sufficient positions for a sign (if one is used), a digit to the left of the decimal point, and a decimal point. It is illegal to format a number as F6.6 with the entire number following the decimal point.

c. On input, valid characters are: spaces or blanks, one decimal point per field, and the numerics Ø-9. Preceding the number, a positive (+) or a negative (-) sign is also acceptable. If no sign is present, the field is assumed to be positive. Spaces or blanks are interpreted as zeros. If the decimal point is present in the input field, it overrides the "d" in the specifications when they contradict one another. If the decimal point is not present, then the editing code "d" determines where it will be placed.

d. If the entire field is blank, a value of -0.0 is stored in the memory.

e. On output, if the field is negative a negative sign (-) will be printed preceding the most significant digit. If the value is positive, no sign will appear. If the decimal portion of the number has fewer digits than the specification requires, it will be filled on the right with zeros. If the entire field width is larger than necessary, it will be filled on the left with blanks. If the field width is not large enough to contain the number including the negative sign, the field will be filled with asterisks (*) to indicate overflow.

f. When the scale factor is used with the F format for input, the input value is multiplied by 10^{-n} during conversion. For example, a scale factor of one will cause a card value of 2.367 to be stored in the computer as .2367.

5.4.5.2. Fw.d (Continued)

e. (Continued)

Examples of F type input data with format F6.2:

␣12.17

␣1217

+12.17

-12.17

+1217

-1217

+1.217

␣1.217

The following examples show how each of the quantities on the left is printed according to the format code F5.2:

Internal Value	Printed Value
12.17	12.17
-41.16	*****
-.2	-0.20
7.3542	␣7.35
-1.	-1.00
9.03	␣9.03
187.64	*****
. 0	0.00

5.4.5.3. Ew.d

a. "E" specifies conversion between an internal real value and an external number written with an exponent. The total number of characters in the external medium is w, including signs, decimal point, E, exponent and any blanks.

b. On input the use of an actual decimal point is optional; if it is supplied in the field, it overrides d. The exponent part of the field takes the general form E±ee, as in a floating point constant in a statement. However, several shortcuts are permitted. A positive exponent may appear with the + omitted or replaced with a blank, that is, E ee or Eee. If the first digit of the exponent is zero, it may be omitted. If the exponent is written with a sign, the E may be omitted. Thus all of the following are permissible and equivalent forms for the exponent plus 2:

E+02,E02,E 02,E+2,E2,+2

The exponent must be at the extreme right of a field to convey the correct meaning.

All of the following data fields convert to the same internal number if read is under control of E14.7, but it must be remembered that an actual decimal point overrides d in the field specification:

```
+ 1.2345678E03
    ^
  12345678.E-4
     ^
    1234.5678E0
       ^
+ 0.12345678+4
     ^
```

c. On output the number will normally appear in the form ±.nn ....E+ee (except that plus signs are replaced with blanks), in which the number of places after the decimal point is specified by d. A scale factor has no effect on input with an E specification. On output the effect is to multiply the "fractional" part by 10^s and to reduce the exponent by s. The scale factor may be positive or negative. If the value is zero, no exponent will appear.

The following examples show how each of the quantities on the left (the internally stored real number) is printed according to the format code E8.3:

INTERNAL VALUE	PRINTED VALUE
1567.23	.156E04
-42.35	.423E02
6.3356	.634E01
7.1	.710E01
-.2	-.200E00
-25.3	-.253E02

5.4.5.4. Gw.d

"G" is a generalized real conversion which, on output (WRITE), includes the capabilities of both E and F conversion codes. On input (READ), a Gw.d is the same as Fw.d.

On output, the code Gw.d attempts to transmit a real datum of absolute magnitude N in a Fw.d mode, as per the following rules:

If the absolute magnitude of N is	then the effective output code is
$0.1 \leq N < 1$	F(w-4).d,4X
$1.0 \leq N < 10$	F(w-d).(d-1),4X
.	
.	
.	
$10^{d-2} \leq N < 10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N < 10^{d}$	F(w-4).0,4X

If the absolute value of N is outside this range, the effective editing code is Ew.d.

For example: The editing code is G12.5 and the item is +125346.1. The item falls outside the limit set by 10^5 and therefore will be printed with the code E12.5, resulting in ⁄⁄.12535E+06. If the editing code had been G14.7, the effective editing code would have been F10.1,4X resulting in the number ⁄⁄125346.1⁄⁄⁄⁄.

To insure correct conversion to the Ew.d format when required, the "w" of the Gw.d editing code must be at least 7+d for negative numbers. Also note that the effect of a scale factor is suspended when the absolute magnitude of the number falls outside the range shown in the table.

5.4.5.5. Dw.d

Dw.d is described as Ew.d except that the list item is regarded as DOUBLE PRECISION. (Two locations are taken.) See the description of a DOUBLE PRECISION constant.

Example: D16.4

5.4.5.6. FIELD DESCRIPTOR FOR A COMPLEX NUMBER

A complex value is represented by an ordered pair of real numbers. Therefore the rules described for handling real value using the E, F, and G conversions apply. The only difference is that one name appears in the name list of a basic statement for each pair of numbers; each member of the pair requires a field descriptor.

Example: 2F10.5

5.4.5.7. Ow

This specification indicates that the field is to occupy w print positions, that the value of the list item is to be interpreted as a 12-digit octal number, and that the quantity is to be printed as an octal number, right-justified in the field. If the field width is less than or equal to 12, then the w least significant digits will appear. If the field width is larger than 12, then it will be filled out to the left with blanks. Leading zeros will be printed if the field width allows. The "O" specification is considered typeless; therefore, any type list may be used.

Example 1: Three fields are to be printed.

```
10   FORMAT (3O12)

     WRITE (6,10)(A(I),I = 1,3)
```

Example 2: Three fields are to be read.

```
20   FORMAT (O6,O4,O2)

     READ (5,20) (N(I),I = 1,3)
```

The three cells of N now contain leading zeros and the number of octal characters described in the format.

5.4.5.8. Lw

The "L" phrase specifies that the data will be a logical constant. If w is specified for output and the value of the logical list item is .TRUE., the rightmost position of the field with length w will contain the letter "T". The "F" is printed instead if the value is .FALSE.. On input, the field width is scanned from left to right for the first letter "T" or "F" and the value of the corresponding logical list item is set to be .TRUE. or .FALSE., respectively. All other characters of the external input field are ignored.

5.4.5.9. wH

For input or output of alphameric information, two kinds of specifications are available: One which keeps the alphameric information only in the FORMAT declaration's outside list, called the Hollerith specification, and another which enables a variable name to be given to the alphameric information, by means of the corresponding inside list, called the A specification.

The general format for the Hollerith specification is

```
wHXXX...
```

where w is the integer number of characters which immediately follow the "H", and the "X's" denote the characters which form the information.

For example:

```
1Ø FORMAT  (32H THIS IS ALPHAMERIC INFORMATION.)
```

The few blanks and the period are counted giving a total of 32 characters.

On input, it is assumed that there already exists old alphameric information, and the number of characters specified will be read into the FORMAT list to replace the old information exactly. Thus, for example, if on a punch card appears

```
△THIS△IS△NEW△INPUT△INFORMATION.
```

and we had

```
   10  FORMAT (32H THIS IS ALPHAMERIC INFORMATION.)
       READ(5,10)
```

5.4.5.9. wH (Continued)

the result would be to change the format declaration 10 to

10 FORMAT (32HΔTHIS IS NEW INPUT INFORMATION.Δ)

for some future WRITE or other output statement. In this way, headings on tables can be changed for successive printouts.

When this edit code is used for output, the editing code is independent of the variable list in a corresponding WRITE statement. The characters following the H are picked up and become a part of the output record.

The first character should be a carriage control character for the printer and is not printed. The following table indicates the effect of the carriage control:

CHARACTER 1	ACTION BEFORE PRINTING
Blank	Single Space
Ø (ZERO)	Double Space
1	Skip to New Page
+	Suppress Spacing

Note:

An additional feature in FORTRAN V is the use of single quotes to surround the data thus replacing the wH and avoiding the necessity to count characters.

5.4.5.10. Aw

The "A" format phrase specifies that the next w characters of data are to be interpreted as a string of Hollerith characters. Differing from the "H" format phrase, the information itself is stored in an address and hence has a name in the corresponding inside list. The name of the alphameric variable may be of any type. However, if there is to be any manipulation of these data in the program, the arithmetic used in the manipulation will be of the type specified by the variable's name. Since only six characters can be stored in a single word, the number of characters must not be greater than six. If a larger string of characters is required, then successive specifications must be used in the outside list, and the inside list of names for the information can be made a subscripted variable. For example, suppose we desired to read in

Δ THIS Δ IS Δ ALPHAMERIC Δ INFORMATION

the FORMAT declaration would be

```
10 FORMAT (A6,A6,A6,A6,A6,A2)
```

or

```
10 FORMAT (5A6,A2)
```

Similarly, for output, the information in the contents of the corresponding addresses of the inside list is printed out. For example, if the PRINT statement were

```
WRITE(6,10) (ALPHA(I), I=1 ,6)
```

then the contents of the first six addresses of the array ALPHA would be printed out as alphameric information.

In input, if the total field width is equal to 6, then the 6 characters will be picked up and put into one computer word. If the total field width is greater than 6, then the rightmost 6 characters will be picked up and put into one computer word. If the total field width is less than 6, then "w" characters will be picked up and left-justified in one computer word with Fieldata space fillers on the right.

On output, if the total field width is equal to 6, the 6 characters in memory will be printed. If the total field width is greater than 6, then the 6 characters in memory will be right-justified and printed, with spaces filling the leftmost areas. If the total field width is less than 6, then the "w" leftmost characters in memory will be printed.

If the list item is DOUBLE PRECISION, or COMPLEX, a descriptor A12 may be used in order to obtain a field of length 12.

5.4.5.11. wX

This specification indicates that a field, w in length, is to be filled with blanks (on output) or ignored (on input). This specification is independent of the list items. This feature is used to skip characters on input and to insert spaces on output.

5.4.5.12. SPECIAL NOTES ON REAL VALUE DESCRIPTORS E,F, and G

These three conversion codes do not distinguish between external representations of real numbers for input of data. On the formatted READ, any of the following data representations are acceptable to the conversion codes E, F, and G.

12.34
1245
34E2
37+2
95.6E5
57.6D8

For output of data, the E, F, and G conversion codes indicate a unique external form.

5.5. TAPE POSITIONING STATEMENTS

There are three statements specifically for use with magnetic tape. In each case the verb is followed by an integer constant or variable designating the FORTRAN unit number.

5.5.1. END FILE t

This statement places an end-of-file mark on the designated tape. Its primary use, from our standpoint, is to serve as a signal to the off-line printer that no more valid information follows.

5.5.2. REWIND t

This statement returns a tape to its beginning. This step is necessary, for instance, when a binary tape has been written with intermediate results and we want to read the data back into the computer. Tape cannot be read backward in FORTRAN, so we rewind the tape and use a binary read to get the information. A tape should be rewound each time before use because we cannot know for sure that the tape is at its initial point, even though it ordinarily will be. If by chance it is positioned somewhere else, we will have no trouble in writing, but when we later rewind the tape and try to read the information we have written we will encounter tape area before the point at which we began writing. Usually, there will be information on it from some previous usage of the tape, and we will be in trouble.

5.5.3. BACKSPACE t

This statement backs up the specified tape by one "record." A record, for our purposes, may be defined as the information corresponding to one card or one printer line.

5.6. UNFORMATTED INPUT AND OUTPUT

When a given FORTRAN program writes a file to be processed by another FORTRAN program, it is sometimes advantageous to use the unformatted WRITE and READ. The units and lists specified conform in all respects to those discussed for formatted WRITE and READ statements. However, unformatted data appears on the external medium in an unedited form. That is, no conversion takes place. The entire list of variables, arrays, and array elements given in an unformatted WRITE statement is transmitted to an external device as a single logical record. It exits as a single input record for an unformatted READ statement. Tape unit numbers -1, -2, -3, 5 or 6 should not be used with unformatted reads. The forms of the unformatted READ or WRITE are:

```
READ (unit) list
WRITE (unit) list
```

5.7. PRINTING A FORMATTED TAPE

Because of the system used at Census, printing is not done on-line at the time a program is executed. Instead, a service routine is used to print an output tape in lieu of direct printing. CENSER, option TP (tape-to-print function), is used. If the data to be printed exceeds 100 blocks, then it is requested that the FORTRAN formatted write not be used. The FOLION subroutines and Census functions should be used, in that instance, to prepare a tape for the Census off-line high-speed printers, which require BCD-XS3 characters. However, a service routine (EDTCPY) is available when necessary to translate and copy a Fortran formatted tape to a IIA tape which can be printed off-line using Board 3, Loop 4, single-space.

5.8. EXERCISES

5.8.1. When using a FORMAT statement, which type of the following merely moves the data into an area without changing the character(s)?

F, E, G, A, I

5.8.2. Show how the given data values would be printed under control of the field specifications stated:

a. I5 : 0, 10, -587, 90062, 123456
b. F7.2 : 0.0, 1.0, 16.77, -586.21, 0.04, 12.34
c. F5.0 : 0.0, 1.0, 16.87, -12.32
d. E10.3 : 0.0, 0.00072, 601000., -473., -0.0123
e. 1PE10.3: 0.0, 10.0, 0.000076, 6780000., -627, -0.000456

5.8.3. Given that the values of three variables are as follows,
M = 12, X = 407.8, Y = -32.9
Show exactly what would be printed by

```
WRITE (6,107) M, X, Y
```

with each of the following FORMAT statements:

a.
```
107 FORMAT(1H␣,3HM =␣,I3,3HX =␣,F6.1,3HY =␣,F6.1)
```

b.
```
107 FORMAT
   A(1H1,17HREADING NUMBER = ,
   B I3/1HO,8HPRESSURE,6X,
   C 11HTEMPERATURE/1HO,
   D F7.1,9X,F6.1)
```

5.8.4. Four numbers are punched on a card; they are new values of real variables named BOS, EWR, PHL, and DCA. Each number is punched in 8 columns, the first beginning in column 1. Each number contains a decimal point. Write READ and FORMAT statements to read the card.

5.8.5. Repeat exercise 5.8.4., but assume that there is no decimal point in the data. Treat the numbers as if there were two decimal places.

5.8.6. Repeat exercise 5.8.4., but let each field occupy 14 columns and a decimal point and an exponent appear in the data.

5.8.7. Given a WRITE statement,

```
WRITE (6,92) I,J,R,S
```

I and J are integer variables and R and S are real.
For each of the following, write a FORMAT statement that could produce a line or lines with the given WRITE statement.

a. ␣␣␣␣-16␣␣92017␣␣␣16.82␣␣437.89
b. ␣␣␣␣-16␣␣92017␣␣␣17.␣␣438.
c. ␣␣␣-16␣␣␣92017␣␣0.16824E␣02␣␣0.43789E␣03
d. ␣␣␣-16␣␣␣92017␣␣1.6824E␣01␣␣4.3789E␣02
e. I=␣␣␣-16␣␣J=␣92017␣␣R=␣␣16.8␣␣S=␣437.9
f.
```
I=␣␣␣-16
J=␣92017
R=␣␣16.8
S=␣437.9
```

5.8.8. DATA is a one-dimensional array of at most 10 elements. A card is punched with a value of N in columns 1 and 2 and with 1 to 10 elements of DATA in succeeding columns. The number of elements is given by the value of N. Each number is punched with a decimal point but no exponent, in seven columns. Write statements to read such a card.

5.8.9. Describe in words what card format and deck makeup would be required for each of the following groups of statements to be meaningful:

a.
```
      DIMENSION X(10)
      READ(5,69) (X(I),I=1,7)
   69 FORMAT(10F4.0)
```

b.
```
      DIMENSION X(10)
      READ(5,70) N, (X,(I), I=1,N)
   70 FORMAT(I2,10F4.0)
```

c.
```
      DIMENSION X(10)
      READ(5,71) N, (X(I), I=1,N)
   71 FORMAT (I2/10F4.0)
```

d.
```
      DIMENSION X(10)
      READ(5,72) N, (X(I),I=1,N)
   72 FORMAT (I2/(F4.0))
```

e.
```
      DIMENSION Y(10,10)
      READ (5,79) K, (Y(K,I),I=1,10)
   79 FORMAT(I2,10F5.0)
```

f.
```
      DIMENSION Y(10,10)
      READ(5,80) K,M, (Y(K,I),I=1,M)
   80 FORMAT(2I2,10F5.0)
```

g.
```
      DIMENSION Y(10,10)
      READ(5,81) M,N, ((Y(I,J),J-1,N), I-1,M)
   81 FORMAT(2I2/(10F8.0))
```

h.
```
      DIMENSION Y(10,10)
   83 READ(5,82) I,J,Y(I,J),L
   82 FORMAT(2I2,F10.0,I1)
      IF (L.EQ.0) GO TO 83
```

5.8.10. Express the following numbers using the E format notation:

a. 10.00
b. .00000492
c. 100

5.8.11. Describe the contents of the printing resulting from:

```
   12 FORMAT  (1X,4I10/1X,40I1)
      WRITE(6,12) (II,II = 1,4), ((II,II = 1,10), JJ = 1,4)
```

5.8.12. Calculate the sample mean, variance, and standard deviation for n values (up to 250) punched one per card in the form XXX.X beginning in column 1. Let the first card contain the number of values which follow. Print the results. Use the formula:

$$\text{Variance} = \frac{\sum_{i=1}^{n} X_i^2 - \left(\sum_{i=1}^{n} X_i\right)^2}{n\,(n-1)}$$

The standard deviation is the square root of the variance.

The sample mean is the sum of the values X_i divided by the number of values.

5.8.13. Prepare a program to compute simple interest for several years. Read in values of principal, interest rate, and an upper limit to the period. Calculate the amount at the end of each period from 1 to the limit, printing out values of the period and the amount. Prepare the following table.

```
PRINCIPAL   XXXXXXX.XX        RATE   XXX.XXX

PERIOD          AMOUNT
   X      XXXXXXXX.XX
   "              "
   "              "
   "              "
```

5.8. ANSWERS TO EXERCISES

5.8.1. A

5.8.2. a. 0, 10, -587, 90062, *****
b. 0.00, 1.00, 16.77, -586.21, 0.04, 12.34
c. 0., 1., 17., -12.
d. 0.000E00, .072E-02,.601E06,-.473E03,-.123E-01
e. 0.000E00, 1.000E01,.760E-04,6.780E06,-6.270E02,-4.560E-04

5.8.3. a. M = 12X = 407.8Y = -32.9

b. READING NUMBER = 12

PRESSURE	TEMPERATURE
407.8	-32.9

5.8.4.
```
      READ(5,50) BOS, EWR, PHL, DCA

   50 FORMAT (4F8.0)
```

5.8.5.
```
      READ(5,51) BOS, EWR, PHL, DCA
   51 FORMAT (4F8.2)
```

5.8.6.
```
      READ(5,60) BOS, EWR, PHL, DCA
   60 FORMAT (4E14.0)
```

5.8.7. a.
```
      FORMAT (2I7, 2F8.2)
```
b.
```
      FORMAT (2I7, 2F6.0)
```
c.
```
      FORMAT (I6,I8, 2E13.5)
```
d.
```
      FORMAT (I6,I8, 1P2E12.4)
```
e.
```
      FORMAT (3HbbI=, I6, 4HbbbJ= ,I6,
     A 4HbbbR= ,F6.1, 4HbbbS= ,F6.1)
```
f.
```
      FORMAT (3HbbI=, I6/3HbbJ=, I6/
     A 3HbbR=, F6.1/3HbbS=, F6.1)
```

5.8.8.
```
      DIMENSION DATA(10)
      READ(5,80) N, (X(I),I-1,N)
   80 FORMAT (I2,10F7.0)
```

5.8.9. a. Only one card will be read. Seven fields of data, four characters each, will be read. If no decimal points appear the number will be considered integer and converted to real. If a decimal point appears, it takes effect.

b. One data card will be read. The first field of 2 digits should contain a number which will be used as an index to read in the numbers which appear in the card. Each number occupies 4 columns.

5.8.9. ANSWERS TO EXERCISES (Continued)

5.8.9. (continued)

c. Two cards will be read. The first card should contain an integer in columns 1-2. The second card should contain as many 4-digit numbers as the value of the integer in the first card.

d. The first card read will contain a number which describes the number of cards to follow. Each card will contain one four-digit number.

e. One card will be read. The first two digits contain a number which will be used as the first subscript of the array Y. The second subscript of the array Y will begin at 1 and end at 10. Thus the 10 values read in will be placed into the array Y.

f. One card will be read. The fields K and M are given values from the first two two-digit fields read in. Then K becomes the first subscript of the array Y. The second subscript I varies from 1 until it reaches the value of M. Thus M values are read into array Y.

g. The first card contains values for M and N; N becomes the limit for the second subscript which begins at 1. M becomes the limit for the first subscript which begins at 1. As many values as necessary are read to satisfy the requirements of the limits of the subscripts. Each card will furnish 10 values. The J subscript varies the most rapidly.

h. Each card contains four numbers. The first two fields placed into I and J specify the subscripts to be used to determine the address within the array Y to which the third field will be sent. The fourth field goes to L. In the event that the integer L is zero, the next card is read.

5.8.10.
a. .1E02
b. .492E-05
c. .1E03

5.8.11.
Δ 1 2 3 4

Δ123456789*123456789*123456789*123456789*

5.8.12. (ANSWER)

```
C....    SAMPLE MEAN, VARIANCE, STANDARD DEVIATION
         DIMENSION X (250)
    3    FORMAT (I3)
    4    FORMAT (F5.1)
         SUMX = 0.0
         SUMSQ = 0.0
         READ (5,3) N
         DO 100  I = 1,N
         READ (5,4) X (I)
         SUMX = SUMX + X (I)
         SUMSQ = SUMSQ + X (I) **2
  100    CONTINUE
         ZN = N
         XBAR = SUMX/ZN
         VAR = (ZN * SUMSQ - SUMX**2)/(ZN*(ZN-1.0))
         STDEV = SQRT (VAR)
         WRITE (6,5) XBAR, VAR, STDEV
    5    FORMAT (1H1, 3F12.2)
         STOP
         END
```

5.8.13. (ANSWER)

```
C...     SIMPLE INTEREST FOR SEVERAL YEARS
    1    FORMAT (F7.2, F5.3, I2)
    2    FORMAT ( 10H1PRINCIPAL, F10.2, 6HΔΔRATE, F8.3)
    3    FORMAT (15HΔPERIOD ΔΔAMOUNT)
    4    FORMAT (I5, F12.2)
         READ (5, 1)  P, RATE, NL
         WRITE (6, 2) P, RATE
         WRITE (6, 3)
         DO  10  N = 1, NL
         ZN = N
         A = P * (1.0 + ZN*RATE)
         WRITE (6, 4) N, A
   10    CONTINUE
         STOP
         END
```

6. FUNCTION AND SUBROUTINE PROCEDURES

6.1. GENERAL

At various points in a program, operations may be required which are identical, or almost identical to others (i.e., the operations are the same but the data values are different). FORTRAN provides a method of writing these operations once in a general form, and making them available as many times as necessary in a program. Each such group of statements is called a procedure or subprogram. Each procedure has a unique name by which it is referenced in a program. Thus, by merely designating a name, procedures are repeatedly executed by the program without the programer having to duplicate the statements comprising the procedure.

As the FORTRAN language evolved, two basic kinds of procedures developed -- functions and subroutines.

There are three basic differences between a function procedure and a subroutine procedure:

FUNCTION	SUBROUTINE
A function requires at least one argument (datum to be operated on).	A subroutine may, or may not have an argument(s).
A function reference (the function name) appears as an expression or as a term of an expression.	A subroutine reference is an independent statement. See CALL statement.
A function procedure returns a single value to the point in an expression which referenced it.	A subroutine procedure may, or may not, return information to the referencing program.

6.2. PROCEDURE ARGUMENTS

Arguments are data specified in the procedure reference which are passed on to, or used in, the computational procedure. An argument may be a name, an expression, or a Hollerith constant. The arguments are acted upon by the procedure to produce one or more results. The arguments, in list form, must be separated by commas, and the entire argument list enclosed in parentheses, e.g., (X, Y, ADD, MORE).

Arguments in the procedure are referred to as either dummy or formal arguments. These arguments, consisting of symbolic names, represent the data values or procedure names which can change with each procedure reference made. The arguments to be transferred from the referencing statement to the procedure are called actual arguments and the formal or dummy arguments are those mentioned in the procedure. The actual and the dummy arguments do not have to use the same names, but the actual arguments must correspond in number, data type, and order to the dummy arguments stated in the procedure. When the machine language program is executed, the dummy arguments are replaced by the actual arguments from the statement making the reference to the procedure.

The following table describes the general forms of actual arguments and gives pertinent examples:

ACTUAL ARGUMENT FORM	EXAMPLE	
	Actual	Dummy
Variable Name	DIV	B
Array Element	ARRAY (1)	X
Array Name *	MATRIX	JDE
Expression	(S**2./3.0)+1	Z
Procedure Name **	SIN	Y
Hollerith Constant	6H TOTAL	HOLL
Statement Label ***	$999	$

* An array name passed on to a procedure requires that the procedure declare the array in a statement such as DIMENSION.

** Procedure names used as actual arguments must be specified in an EXTERNAL statement in the calling program.

*** The "$" as the i^{th} argument becomes the i^{th} RETURN to the location specified in the formal argument.

6.3. THE FUNCTION

A function is a computational procedure which must be supplied with at least one argument from the statement containing the function reference. A function must return a single value to the position in the statement where the reference occurred and may return additional values to be used in subsequent statements of the program.

All functions, whether written by the programer or supplied by the FORTRAN system are referenced in the same way.

Function names conform to the general naming convention.

6.3.1. REFERENCING A FUNCTION

A function is referenced in a program by merely including the function name and the actual arguments in an expression in the same manner that constants and variables are written in an expression. For example, if the sine of a value is needed as part of a computation, the sine function appears in a statement as follows:

```
A = B + SIN (C)
```

where SIN is the symbolic name for the sine function and C is the argument. Arguments for all procedures must be enclosed in parentheses. Thus, the single value of SIN (C) is computed, added to the value of B, and the result placed in the location represented by A.

Additional examples of using functions are as follows:

```
A = ABS (B) + AMAX1 (C, D, E)

A = COS (B + C)

A = SIN (SQRT (B) - CBRT (C))

A = SIN (COS(TAN(SQRT(CBRT(A)))))
```

The expression in the above statement is called a nested function reference. Nested functions are evaluated in the sequence starting with the innermost parenthetical level to the outermost parenthetical level. Thus:

```
Let T1 = CBRT (A), then
    T2 = SQRT (T1) and
    T3 = TAN (T2) and
    T4 = COS (T3) and
    A  = SIN (T4)
```

6.3.1. REFERENCING A FUNCTION (Continued)

There are four kinds of functions available in the FORTRAN language.

1. Built-in function (in-line object code generated by the FORTRAN processor).

2. Statement functions (programer created in one statement).

3. External functions (library function and function subprograms compiled independently).

4. Internal function (a subprogram written by the programer and compiled in conjunction with a main program or another subprogram).

Internal and external functions are both subprograms which are referenced in an expression and return a single value to that expression.

A list of functions generally supplied by FORTRAN are shown in the tables BUILT-IN (Intrinsic) Functions and External Functions. The programer need be concerned with these distinctions, intrinsic and external, only if he uses a name which is the same as that of a supplied function name.

The name of an external function used as the name of a procedure subprogram written by the programer causes the subprogram to replace the supplied procedure. Thus, the programer may write his own sine or tangent procedure if he desires.

In addition to the FORTRAN supplied functions, there is a group of CENSUS-written functions and subroutines which will be described later.

It was found that the functions provided by FORTRAN were insufficient to handle the programer's needs. Therefore, a method of allowing a programer to write function procedures was introduced. The first is the statement function. This allows the programer to write one statement specifying the steps required for the evaluation of a function. For writing a function procedure requiring more than one statement, a function subprogram is used. Any function must specify at least one argument and return a single value to the point of the function reference in the program. For other kinds of required procedures where arguments may or may not be necessary, and where data may or may not be required, a procedure called a subroutine subprogram is written. If the subprogram is compiled independently, it is considered external. If it is included in an assembly as one program unit with only one END card for both the main program and all its subprograms, then it is considered to be internal.

6.3.2. STATEMENT FUNCTIONS

Statement functions are composed by the programer in the following form and they must be prepared ahead of the first executable statement.

$$f\ (P_1,\ P_2,\ \ldots,\ P_m) = e$$

where f is the function name; P_i are distinct, non-subscripted variable names (formal arguments); and e is either an arithmetic expression if the mode of f is arithmetic, or a logical expression if the mode of f is logical. Normally the formal arguments will appear in the expression e. Upon execution of the statement function, they are replaced by the actual arguments of the function reference. They may each be of any mode. The actual arguments of a given reference must agree in number and mode with their corresponding formal arguments. It is not necessary that the formal argument names be distinct from any other names. They are local to the statement function.

The function names must be unique. They must not be the same as any variable name appearing elsewhere in the FORTRAN program in which they appear. A function whose value is logical must be declared LOGICAL.

The expression e may have subscripted variables appearing in it. However, the formal arguments may not be subscripted. The expression e may also contain references to previously defined statement functions, built-in functions or external functions.

References to statement functions may not contain statement labels as arguments (i.e. F ($5)).

The name f of the statement function must not appear in a DIMENSION, EQUIVALENCE or COMMON statement in the main program or subprogram where it is defined.

The type of the statement function is determined by the name convention or by a previous type statement.

6.3.3. EXAMPLES OF STATEMENT FUNCTIONS

1. Non-executable statement function appearing before any executable statement:

```
ROOT (A,B,C) = (-B+SQRT(B**2-4.*AC))/2.*A)

To use the function called ROOT:

  ANS = ROOT (16.9,R-S,T+6.9)+COS(X)
```

2. Two statement functions to illustrate how a second function may reference the first one:

```
S(A,B,C) = (A+B+C)/2

AREA(A,B,C) = SQRT(S(A,B,C)*(S(A,B,C)-A)*
1                 (S(A,B,C)-B)*(S(A,B,C)-C))
```

3. A statement function to compute interest:

```
A(P,R,N) = P*(1+R)**N+.005
```

6.3.4. BUILT-IN FUNCTIONS

The 1107 FORTRAN IV Compiler recognizes 31 names as built-in functions. The compiler will generate in-line code for the evaluation of the built-in function value whenever reference occurs.

The arithmetic expression,

$f(a_1, a_2, \ldots, a_n)$

references the built-in function with the name f and represents the computed value, where a is any arithmetic expression of the mode specified for the arguments of the built-in function and n is the number of arguments. See the table "Built-In (Intrinsic) Functions."

The names for the REAL and INTEGER built-in functions have been selected so that the result of the function will correspond in type according to the naming conventions established for REAL and INTEGER functions. The DOUBLE PRECISION and COMPLEX built-in functions will begin with a D and a C, respectively. Note, however, that DIM is a REAL built-in function.

6.3.4. BUILT-IN FUNCTIONS (Continued)

The name of a built-in function may be used for purposes in a program other than to request the execution of a particular function. The following uses will overrule the definition of a name as a built-in function:

1. The name is used as a simple variable name.

2. The name is used as a statement function name.

3. The name appears in:

   a. Type statement

   b. EXTERNAL statement

   c. DIMENSION statement

   d. COMMON statement

   e. PARAMETER statement

   f. ABNORMAL statement

   g. FUNCTION or SUBROUTINE statement

Note that if a built-in function produces a COMPLEX or DOUBLE PRECISION result, the programer must not enter the name of that built-in function in a type statement, as the name will not be considered that of a built-in function according to the rules of the above paragraph.

If the definition of the built-in function name is overruled in a main program, the built-in function is not available in the entire program unit. The built-in function names are defined as global names. Therefore, the name may be overruled in an internal subprogram and not affect the availability of the function in the main program or in another internal subprogram.

FORTRAN Name	No. of Args.	Function	Mode of Argument	Mode of Function
ABS	1	Determine the absolute value of the argument.	Real	Real
IABS			Integer	Integer
DABS			D-P	D-P
AINT	1	Truncate; eliminate the fractional portion of the argument.	Real	Real
INT			Real	Integer
DINT			D-P	D-P
AMOD	2	Remainder; subtract from the first argument the appropriate integer multiple of the second argument so that the remainder is less than the second argument but non-negative.	Real	Real
MOD			Integer	Integer
AMAX0	≥ 2	Select the largest value.	Integer	Real
AMAX1			Real	Real
MAX0			Integer	Integer
MAX1			Real	Integer
DMAX1			D-P	D-P
AMIN0	≥ 2	Select the smallest value.	Integer	Real
AMIN1			Real	Real
MIN0			Integer	Integer
MIN1			Real	Integer
DMIN1			D-P	D-P
FLOAT	1	Convert from integer to real.	Integer	Real
IFIX	1	Convert from real to integer.	Real	Integer
DBLE	1	Convert from real to double-precision.	Real	D-P
CMPLX	2	Convert two real arguments to one complex number.	Real	Complex
SIGN	2	Replace the algebraic sign of the first argument by that of the second.	Real	Real
ISIGN			Integer	Integer
DSIGN			D-P	D-P
DIM	2	Positive difference; subtract the smaller of the two arguments from the first argument.	Real	Real
IDIM			Integer	Integer
SNGL	1	Obtain the most significant part of a double-precision argument.	D-P	Real
REAL	1	Obtain the real part of a complex argument.	Complex	Real
AIMAG	1	Obtain the imaginary part of a complex argument.	Complex	Real
CONJG	1	Obtain the conjugate of a complex argument.	Complex	Complex

6.3.5. NOTE on MINO, MIN1, AMINO, AMIN1, MAXO, MAX1, AMAXO, AMAX2

These FORTRAN functions are used to determine the minimum or the maximum of several integer or real numbers. All items in the list must be expressions of the same mode. The names of these functions indicate their use. The letters MIN or MAX indicate whether a minimum or maximum will be sought in the list. The character 1 or the character 0 (zero) which follows indicates the mode of the list: 0 (zero) for an integer list; 1 for a real list. If the function name begins with an A, the mode of the function is real. If it begins with an M, the mode of the function is integer. The function produces a maximum or minimum by setting itself equal to the appropriate item in the list.

CAUTION: The functions which end in 0 (zero) often cause keypunch errors. The last character of MINO, AMINO, MAXO, and AMAXO is a numeric zero (not the letter O).

EXAMPLE 1:

If the largest of three integer variables I, J, K is greater than 50, call subroutine x.

```
IF (MAXO (I,J,K).GT.50) CALL X
```

EXAMPLE 2:

Multiply the variable KET by the whole number portion of the maximum of the real variables A and B.

```
N = KET * MAX1(A,B)
```

6.3.6. NOTE on MOD, AMOD FUNCTION

These FORTRAN functions are used to determine the remainder when one number is divided by the other. MOD is for integer division and AMOD is for real division.

General Form
MOD (K,L)
AMOD (A,B)
K is an integer expression which is to be divided by the integer expression L
A is a real expression which is to be divided by a real expression B

After dividing the first argument by the second, the function sets itself equal to the remainder.

EXAMPLE:

The counter KPAGE is the count of pages on a report. Determine whether the current value of KPAGE is even or odd. If even go to the statement 55.

```
IF (MOD(KPAGE,2).EQ.0) GO TO 55
```

6.3.7. BUILT-IN TYPELESS FUNCTIONS AND, OR, LOC, COMPL, BOOL

6.3.7.1. AND

This sets the function expression equal to a value obtained by a "logical AND" operation of two numbers.

Example:

```
NC = NA.AND.NB

     (Bit in NA)  (Bit in NB)  (Result in NC)

         0            0             0
         0            1             0
         1            0             0
         1            1             1
```

If NA = 010203040506

and NB = 777700770000

then NC = 010200040000

The use of the built-in function is faster than the equivalent CENSUS "LAND". See 8.8.

6.3.7.2. OR

This sets the function expression equal to a value obtained by a "logical addition," or an "inclusive OR" operation.

Example:

```
NC = NA.OR.NB

     (Bit in NA)  (Bit in NB)  (Result in NC)

         0            0             0
         0            1             1
         1            0             1
         1            1             1
```

If NA = 252525252525

and NB = 000077630077

then NC = 252577672577

The use of the built-in function is faster than the equivalent CENSUS "LADD". See 8.6.

6.3.7.3. LOC

This routine returns the location of a function argument as the value. This function may also be referenced through the alternate entry point XLOC.

Example:

```
N = LOC(NB)-LOC(NA)
```

This example will return a value which is the actual number of core locations between the two addresses.

6.3.7.4. COMPL

This routine complements the number found at the address of the argument.

Example:

```
N = COMPL(N)
```

6.3.7.5. BOOL

This routine returns the function argument as the value of the function. No conversion takes place.

Example:

```
M = BOOL(N)
```

6.3.8. EXTERNAL AND INTERNAL FUNCTIONS

Internal and external functions are subprograms, whole sections of coding which return single values to the point in the expressions which reference them.

External Functions

There are two kinds of external functions: (1) library functions, and (2) external functions and subprograms. These will be discussed under the heading SUBPROGRAMS.

The external functions supplied by FORTRAN are listed in the following table. These mathematical functions may be referenced in the general way.

$$f(a_1, a_2)$$

The external functions supplied in the Census library will be discussed in a separate section.

FORTRAN Name	No. of Args.	Function and Definition		Mode of Argument	Mode of Function
SIN	1	Trigonometric Sine:	SIN (X)	Real	Real
DSIN			DSIN (X)	D-P	D-P
CSIN			CSIN (X)	Complex	Complex
COS	1	Trigonometric Cosine:	COS (X)	Real	Real
DCOS			DCOS (X)	D-P	D-P
CCOS			CCOS (X)	Complex	Complex
TAN	1	Trigonometric Tangent:	TAN (X)	Real	Real
DTAN			DTAN (X)	D-P	D-P
CTAN			CTAN (X)	Complex	Complex
ASIN	1	Trigonometric Arcsine:	ASIN (X)	Real	Real
DASIN			DASIN (X)	D-P	D-P
ACOS	1	Trigonometric Arccosine:	ACOS (X)	Real	Real
DACOS			DACOS (X)	D-P	D-P
ATAN	1	Triginometric Arctangent:	ATAN (X)	Real	Real
DATAN			DATAN (X)	D-P	D-P
ATAN2	2		ATAN(X_1, X_2)	Real	Real
DATAN2	2		DATAN (X_1, X_2)	D-P	D-P
SINH	1	Hyperbolic Sine:	SINH (X)	Real	Real
DSINH			DSINH (X)	D-P	D-P
CSINH			CSINH (X)	Complex	Complex
COSH	1	Hyperbolic Cosine:	COSH (X)	Real	Real
DCOSH			DCOSH (X)	D-P	D-P
CCOSH			CCOSH (X)	Complex	Complex
TANH	1	Hyperbolic Tangent	TANH (X)	Real	Real
DTANH			DTANH (X)	D-P	D-P
CTANH			CTANH (X)	Complex	Complex
EXP	1	Exponential (e^x):	EXP (X)	Real	Real
DEXP			DEXP (X)	D-P	D-P
CEXP			CEXP (X)	Complex	Complex
ALOG	1	Natural Logarithm ($LOG_e x$):	ALOG (X)	Real	Real
DLOG			DLOG (X)	D-P	D-P
CLOG			CLOG (X)	Complex	Complex
ALOG10	1	Common Logarithm ($LOG_{10} x$):	ALOG10 (X)	Real	Real
DLOG10			DLOG10 (X)	D-P	D-P
SQRT	1	Square Root $(X)^{1/2}$:	SQRT (X)	Real	Real
DSQRT			DSQRT (X)	D-P	D-P
CSQRT			CSQRT (X)	Complex	Complex
CBRT	1	Cube Root $(X)^{1/3}$:	CBRT (X)	Real	Real
DCBRT			DCBRT (X)	D-P	D-P
CCBRT			CCBRT (X)	Complex	Complex
CABS	1	Determine the absolute value of the argument: CABS (X) For $X = a + ib$	$X = (a^2 + b^2)^{1/2}$	Complex	Real
DMOD	2	Remainder: subtract from the first argument the appropriate integer multiple of the second argument so that the remainder is less than the second argument but non-negative.	DMOD (X_1, X_2)	D-P	D-P

Note: If the result of the Function is Double-Precision or Complex, the Function name must be declared in a Type statement.

6.3.9. NOTE ON SIN, DSIN, CSIN

The variable used as the argument for a trigonometric sine must be expressed in radians. (One radian = 57.2957795131 degrees.) The following statement functions may be used for converting degrees to radians and vice versa:

Radian to degrees:

```
RADDEG(X) = (90.0/ASIN(1.0))*X
```

Degrees to radian:

```
DEGRAD(X) = X*ASIN(1.0)/90.0
```

EXAMPLE:

Find the sine S of a real variable A expressed in degrees.

```
DEGRAD(X) = X*ASIN(1.0)/90.0
.
.
.
.
S = SIN (DEGRAD(A))
```

6.3.10. SUBPROGRAMS

6.3.10.1. EXTERNAL SUBPROGRAM - GENERAL

Within the FORTRAN system, it is possible to define subprograms which may be called upon by other FORTRAN programs. Generally, they are FORTRAN source language programs that cannot be defined by a single statement and are not commonly enough used to warrant their inclusion in a library. External subprograms must be independently compiled and incorporated into other FORTRAN programs for execution. They are not independently executable as a general rule. EXTERNAL subprograms are written behind the END statement of the main program and each external subprogram must have its own FOR card and its own END card. All data references are local.

When COMMON statements are used in both the main program and the subprogram, then the same area can be referenced by both. See Chapter 7.

The function and subprogram statements allow the programer to divide large programs into smaller subprograms or subroutines. This technique is a great aid to the programer in the all important process of removing programing errors (debugging). Program corrections may be made and programs expanded or reduced in scope by processing the appropriate subprogram rather than the entire program. Frequently this results in a considerable saving of computer time.

An external subprogram may be referenced from any part of any other program unit. It must not be referenced from its own program unit. The name of a subprogram may appear in an argument list of a reference only if the subprogram is accessible from the place where the reference occurs. For instance, a reference to an external function is made from a main program which includes an internal subroutine S. If the name S appears in the reference list, then S will be the subroutine referenced when the statement "CALL P" is executed within the external function if "P" is the dummy argument.

Programs over 700 statements in length should be divided into a main program and external subroutine/s to prevent possible overflow of assembly tables. See 16.2.

6.3.10.2. FUNCTION SUBPROGRAMS - (External)

The first statement of a FUNCTION Subprogram must be of the form

type FUNCTION f (a_1, a_2, ..., a_n)

where type is either REAL, INTEGER, LOGICAL, DOUBLE-PRECISION, COMPLEX, or absent, f is a function name and a_i are the formal arguments. a_i may be array names, non-subscripted variable names, or the name of other FUNCTION Subprograms. Each name must be distinct. Any a_i may be the character, $. When the $ character is used as the i^{th} parameter, the i^{th} parameter is never referenced in the function f except in the statement:

RETURN k

If type is absent then the mode of f is determined by a type statement or by the alphabetic naming conventions. If an a_i, a formal argument, is an array name, then that array name must appear in a DIMENSION statement of the subprogram in which it occurs prior to any reference by an executable statement or by a statement function definition. The DIMENSION statement may, only in this case of subprogram usage with formal subscripted arguments, assume a special form. Normally, the only entries that may be made in the subscript positions of an array named in a DIMENSION statement are integer constants, and these integer constants specify the maximum dimensions of the named array. It is possible in FUNCTION or SUBROUTINE subprograms to have maximum dimensions of arrays appear as nonsubscripted integer variables in DIMENSION statements. In this case, the array name and all of the variable subscript names must be listed as formal arguments of the subprogram. This facility enables the subprogram to be made object time variable in its requirements for data storage. The program which contains reference to such a subprogram must also contain a DIMENSION statement which specifies the actual dimensions of such arrays. The nonsubscripted integer variables which appear in such a subprogram as those formal arguments specifying the maximum dimensions of an array can neither appear on the left of an arithmetic or logical statement, nor can they be changed in value in any other way by the subprogram.

All the formal argument names must occur in at least one executable statement in the subprogram. Any actual argument for a FUNCTION subprogram reference may be either an arithmetic expression, a logical expression, an array name or another function or subroutine, which also must appear in an EXTERNAL statement in the referencing program.

The name of a function itself must not appear in a DIMENSION, or statement in the subprogram. However, the name must appear

6.3.10.2. FUNCTION SUBPROGRAMS - (External) (Continued)

at least once on the left side of a logical or an arithmetic statement, as an element of an input list or as an argument or a subroutine call. (See RETURN below.)

The following are permissible arguments of a FUNCTION Subprogram reference:

a. Any arithmetic expression.

b. Any logical expression.

c. An array name.

d. The name of another function or subroutine which also must appear in an EXTERNAL Statement in the calling program unless the name appears as an explicit reference elsewhere in the program.

e. A statement number preceded by the character $.

f. The form:

nHhhh...h

where n is any unsigned integer, and hhhh...h is a string of any n alphanumeric or special characters including "blank".

Note: If the type of the user-written function is declared in the subprogram, then the type should be declared in the calling program.

Example:

```
      COMPLEX CPLX, S, Y, Z
      .
      Z = CPLX(S,Y)
      .
      .
      ....
C FUNCTION SUBPROGRAM
      COMPLEX FUNCTION CPLX(A,B)
      .
      .
      RETURN
      END
```

6.3.10.3 RETURN

A subprogram will normally contain at least one RETURN Statement. Each such statement marks the logical end of flow for the subprogram and causes, in the case of a FUNCTION Subprogram, a return to the statement in which function reference was imbedded, or in the case of a subroutine, a return to the first Executable Statement following the CALL Statement. It will have the general form:

```
RETURN
```

a Second form of a RETURN statement is:

```
RETURN k
```

where k is an integer constant or integer variable. If this form was used the value of the integer, k, specifies that the return will be made to the i^{th} argument of the referencing statement (presumably the i^{th} argument was a statement number preceded by $). There is no limit to the number of statements specifying a RETURN to an i^{th} argument. There may be more than one statement number argument required by a subprogram. This allows conditional returns to various locations within a program.

If a RETURN statement is not specified then the END statement will serve as a RETURN statement.

A third form of a RETURN statement is:

```
RETURN Ø
```

where Ø is an integer constant, Ø. The execution of the RETURN Ø statement will result in a transfer to the system error program.

The statement END must appear as the last statement of an external function or subprogram.

6.3.10.4. EXAMPLES OF FUNCTION SUBPROGRAMS

```
      FUNCTION MAX (X,N)
C     X = integer array
C     N = size of array (N 2)
C     COMPUTES MAXIMUM VALUE OF ARRAY
      INTEGER X (N)
      MAX = X (1)
      DO 1 I=2,N
      IF (X(I).GT.MAX) MAX = X(I)
1     CONTINUE
      END

      FUNCTION SUM (X,N)
C     X = real array
C     N = size of X
C     Computes sum of elements of specified array
      DIMENSION X (N)
      SUM = 0.0
      DO 1 I=1,N
1     SUM = SUM+X(I)
      END

      FUNCTION SMSQ (X,N)
C     X = real array of size N
C     Computes sum of squares of elements of X
      DIMENSION X(N)
      SMSQ = 0.0
      DO 1 I=1,N
1     SMSQ = SMSQ+ X(I)*X(I)
      END
```

6.4. SUBROUTINE SUBPROGRAM (External)

A SUBROUTINE subprogram has so many points of similarity to a FUNCTION subprogram that the treatment here will consist of citing how it differs from a FUNCTION subprogram.

The major difference is that a SUBROUTINE subprogram may produce many values from one reference whereas a FUNCTION subprogram may produce one. Reference to a SUBROUTINE is made with the special CALL statement described below.

A subroutine is always considered ABNORMAL.

In all other respects, subroutine references and external SUBROUTINE subprograms are the same as function references and EXTERNAL subprograms.

The SUBROUTINE subprogram returns values, if any, only through its arguments (by causing values to be stored in the locations associated with its actual arguments as specified in its CALL statement) or through variables in COMMON blocks. No value is associated with the name of a SUBROUTINE subprogram.

The form of a SUBROUTINE subprogram statement is either:

SUBROUTINE s(a_1, a_2,..., a_n)

or

SUBROUTINE s

where s is a subroutine name and a_i are the n arguments. a_i may be any arithmetic or logical expression, variable name or constant.

Reference to a SUBROUTINE subprogram is made through the use of a special statement which has no other use. This statement is the CALL statement and has the form

CALL s(a_1, a_2, ..., a_n)

or

CALL s

where s is the subroutine name and a_i are the <u>actual</u> arguments. a_i may take on all forms described for FUNCTION subprogram references.

Refer to the section entitled "SORT OPTION CODING" for an example of an external subroutine containing several RETURN statements.

6.4. SUBROUTINE SUBPROGRAM (External) (Continued)

Internal Function And Internal Subroutine Subprograms

Internal subprograms are compiled in conjunction with a Main Program, a SUBROUTINE Subprogram, or a FUNCTION Subprogram. In the latter two cases, the first line of compilation must be a FUNCTION or SUBROUTINE statement.

Subprograms are indicated as internal by either of the following subprogram statements:

1. SUBROUTINE S (a_1, a_2, ..., a_n)
2. FUNCTION F (a_1, a_2, ..., a_n)

where S and F are the subprogram names and a_i are the formal parameters. The same rules apply to the subprogram names and the formal parameters as in the previously defined subprograms.

Internal subprograms are referenced in the same manner as external subprograms; i.e., an internal SUBROUTINE is referenced by a CALL statement; an internal FUNCTION is referenced as other functions are referenced. However, such references and calls must be made within the main program or other internal subprograms contained in the main program.

The compiler will assume that all statements appearing between two internal subprogram statements belong to the first internal subprogram. The occurrence of the Internal FUNCTION or SUBROUTINE Subprogram statements will mark the end of the previous program. The END statement will mark the end of the entire set of programs.

Identifiers (names) which are available in a program unit are referred to as "global". The identifiers which are available only to a particular internal subprogram will be referred to as "local".

The following are global names of a program unit:

1. COMMON block names.
2. Names of internal subprograms.
3. All names including COMMON variable names which appear any place in a program unit's main program or external subprogram except formal argument names.
4. Names of built-in functions.

6.4. SUBROUTINE SUBPROGRAM (External) (Continued)

Internal Function and Internal Subroutine Subprograms (Continued)

The following names are local to an internal subprogram:

1. Formal arguments of an internal subprogram.
2. Statement numbers.
3. Formats.
4. Variable names in type or DIMENSION statements.
5. Variable names which have not appeared elsewhere in the program unit.
6. Variable names appearing in the subprogram's COMMON statements.
7. Statement functions defined in the internal subprogram.

In a particular internal subprogram global names including built-in function names and statement functions may be overridden to be local by their appearance in one of the following statements:

PARAMETER

TYPE

DIMENSION

COMMON

The name of statement functions defined in a subprogram can override a global name.

The EQUIVALENCE statement will not force a variable to be local.

6.4. SUBROUTINE SUBPROGRAM (External) (Continued)

Internal Function and Internal Subroutine Subprograms (Continued)

Formal arguments of an external subprogram must not appear in internal subprograms belonging to the same program unit.

Example:

```
     1   DIMENSION D(50),B(I)
           .
           .
           .
         A = C + D(I)

         GO TO 10
           .
           .
           .
    10   CALL Z (A,20)

         X = A + Y (R+S, $6,U)
           .
           .
           .
     6   ........
           .
           .
           .
         SUBROUTINE Z (W, V)

         DIMENSION D (25)
    10   ......
           .
           .
           .
         CALL Q(D(1), $ 12, D(2))
           .
           .
           .
    11   GO TO 10
           .
           .
           .
    12   V = D (J)/B(J)
           .
           .
           .
         SUBROUTINE Q (X, $, T)
           .
           .
           .
         RETURN 2
           .
           .
           .
         CALL Z (E, F)
           .
           .
           .
         END
```

6.4. SUBROUTINE SUBPROGRAM (External) (Continued)

Internal Function and Internal Subroutine Subprograms (Continued)

The SUBROUTINE Statement, SUBROUTINE Z(W,V) indicates the end of the first program. Subroutine Z is terminated by the SUBROUTINE Statement, SUBROUTINE Q(X, $, T) and SUBROUTINE Q is terminated by the statement RETURN 2 or the END Statement. The END statement also terminates the end of the entire set of programs. The global variables for this program are A, B, C, D, Z, Y, R, S, and U. The local variables in the internal SUBROUTINE Z are W, V, and D. Note that D is also a global variable. The local variable D is a unique variable and not the same as the global variable D, even though they have the same name. The only global variable used in SUBROUTINE Z is the variable B. The statement number 10 is local to SUBROUTINE Z and different from the global statement number 10 in the first program.

The internal subroutine, SUBROUTINE Q (X,$,T) has the local variable X, T, E, and F. If the RETURN 2 statement is executed it will return control to the statement number that is inserted in the second dummy argument. If the END statement is used as the return statement of the internal subroutine, control is transferred to the next executable statement of the referencing program.

6.5. ABNORMAL FUNCTIONS

The definition of a Normal Function is an External Function which has all of the following properties:

a. It has no implied (COMMON) arguments.

b. None of its arguments are function outputs. Its single output is through its name.

c. If a function call with a given set of arguments produces a given result, then all references to the function with all arguments identical with the given set will give the identical result. This excludes functions which, for example, have local variables whose values are saved from reference to reference.

d. The function contains no I/O statements.

An ABNORMAL Function is one which fails to possess one or more of the properties listed above. No attempt will be made to optimize the computation of common subexpressions which contain references to ABNORMAL Functions.

The following rules will apply:

a. If an ABNORMAL statement does not occur in a program, then all function references are considered to be ABNORMAL.

b. If an ABNORMAL statement occurs, then all functions so declared will be assumed to be ABNORMAL, and all other functions will be assumed to be Normal.

c. If one and only one ABNORMAL statement occurs, and that statement lists no function names, then all functions are treated as Normal.

For example:

```
      .
      .
      .
1 ABNORMAL F
      .
      .
      .
2 A = F(X) + F(X)
      .
      .
      .
3 A = 2 * F(X)
      .
      .
      .
```

6.5. ABNORMAL FUNCTIONS (Continued)

In the above example, F is defined in statement number 1 as an ABNORMAL Function. This function might contain statements that rewind or backspace magnetic tape, set hardware indicators, etc. If F(X) backspaces magnetic tape one record, then in statement number 2, the magnetic tape unit would be backspaced two records and the output value of F would be doubled and replace the value in A. In statement number 3, only one record on the magnetic tape unit is backspaced, and the output value of F would be doubled, and replace the value in A. Therefore, these statements do not necessarily have the same result. If F(X) were normal the results of statements 2 and 3 would have been identical.

6.6. EXTERNAL STATEMENT

The EXTERNAL statement, contained in the referencing program, allows the processor to distinguish between a variable name and a procedure name. The general form of this statement is as follows:

EXTERNAL $f_1, f_2, \ldots, f_n$

where the symbols "f" are names of external functions or external subroutines and n is a positive number. If the name of an external subprogram appears in the referencing program unit in reference lists only, the name must also be contained in an EXTERNAL statement. The statement EXTERNAL f in the main program or external subprogram of the calling program unit makes f recognized as an external subprogram in the entire program unit. The same statement in an internal subprogram limits recognition of f as an external subprogram to that internal subprogram. Note that a single function reference $f(a_1,a_2,\ldots, a_n)$ or the statement CALL $s(a_1, a_2,.., a_n)$ placed anywhere in the program unit makes the appearance of f (function) or s (subroutine) respectively in an EXTERNAL statement unnecessary.

The order of the procedure names in an EXTERNAL statement is arbitrary.

6.6. EXTERNAL STATEMENT (Continued)

An example of using EXTERNAL is the case where an argument list of a procedure called COMP requires the (supplied) function SIN, and a procedure written by the programer called SUM. The statement appears as follows:

Referencing Program

```
EXTERNAL SIN, SUM

A = 4.2 + COMP((Y/(2)+X), SIN, SUM)
        .
        .
        .
        .
```

Procedure

```
FUNCTION COMP (XRAY, A, B)

Z = KRAY + .2

COMP = A(Z) - B (Z)
```

Procedure names used only as arguments should appear in the EXTERNAL statement. That is, if the argument list were COMP ((Y/2*X,SIN(D),SUM(E,F)), the values associated with names SIN and SUM would be computed prior to transfer of the arguments. Therefore, SIN and SUM are not declared in an EXTERNAL statement.

6.6. EXTERNAL STATEMENT (Continued)

Example of the Use of the EXTERNAL Statement

As an example of what can be done, the following program contains a main (calling program) and a SUBROUTINE program. The subprogram contains exactly one executable statement: Y = F(X)

The arguments listed are X,F, and Y, making the function F a matter of choice in the subprogram call.

The main program calls this subprogram three times. Each time the value of X is 2.0 and the actual variable corresponding to Y is RESULT. The arguments corresponding to F are successively SIN, COS AND SQRT. These three supplied function names are listed in an EXTERNAL statement. In place of standard mathematical functions, original FUNCTIONS might have been written. Built-in functions may not be used in EXTERNAL statements because they are coded "in line". Examples are: MOD, AMOD, ABS, IABS, FLOAT, and IFIX.

```
      EXTERNAL SIN,COS,SQRT
      CALL SUBR (2.Ø,SIN,RESULT)
      WRITE (6,129) RESULT
129   FORMAT(11HØSIN(2.Ø) = ,F1Ø.6)
      CALL SUBR (2.Ø,COS,RESULT)
      WRITE (6,13Ø) RESULT
13Ø   FORMAT(11HØCOS(2.Ø) = ,F1Ø.6)
      CALL SUBR (2.Ø,SQRT,RESULT)
      WRITE (6,131) RESULT
131   FORMAT(12HØSQRT(2.Ø) = ,F1Ø.6)
      STOP
      END
      ~~~~~
      ~~~~~
      SUBROUTINE SUBR(X,F,Y)
      Y = F(X)
      RETURN
      END
```

6.7. ERRORS TO BE AVOIDED WHEN USING SUBPROGRAMS

Many kinds of errors can occur in the writing and use of subroutines. Most of these errors are errors of omission or errors of inconsistency. The EXTERNAL declaration must appear in the calling program when a procedure name is used as an actual parameter. The dimensions of an array must appear as separate actual parameters when an array appears in a subroutine whose dimensions depend on the calling program. The number and type of actual parameters must agree exactly with the number and type of formal parameters. The actual parameters in a CALL statement must be listed in precisely the same order as the corresponding formal parameters are listed in the procedure heading. When a formal parameter represents an array name, the dimensions must be declared in the subroutine program. The punctuation RETURN must be placed wherever a return to the calling program is to be made from the procedure program.

The name of a procedure itself cannot be used within the program for that procedure. None of the formal parameters can appear in EQUIVALENCE declarations. When writing a FUNCTION subprogram, the value must be returned to the name of the FUNCTION.

6.8. EXERCISES:

6.8.1. Prepare a statement function which could be used to compute one half the perimeter of a triangle.

6.8.2. Prepare a subroutine to check a variable N for the value 0. If the value is not 0, code a return to a statement number in the main program.

ANSWERS:

6.8.1. P(A,B,C) = (A+B+C) / 2.Ø

6.8.2.
```
      SUBROUTINE CHECKZ (N,$)

      IF (N) 1,  ,1
      RETURN
    1 RETURN 2
      END
```

7. SPECIFICATION STATEMENTS

7.1. GENERAL

Specification statements are used to allocate storage space to variables and arrays, to permit sharing of storage space by variables and arrays in one or more programs, and to specify data to be pre-stored in storage locations assigned to variables and arrays.

Four types of specification statements will be included:

1. DIMENSION
2. EQUIVALENCE
3. COMMON
4. DATA

Specification statements are non-executable. All reference of executable statements to a variable appearing in a specification statement must be preceded by that specification statement.

7.2. DIMENSION STATEMENT

This statement is used to declare arrays and has the following general form:

$$\text{DIMENSION } a_1(c_{11},c_{12},\ldots,c_{1j}),\ a_2(c_{21},c_{22},\ldots c_{2k}),\ldots\ldots$$

$$\ldots\ldots,a_n(c_{n1},c_{n2},\ldots\ldots c_{nm})$$

where the a's stand for the names of the arrays. The integers j,k,...,m indicate the number of dimensions of their respective arrays. The number of dimensions may be one to seven. n is a positive integer and the symbols "c" are positive integers or parameter variables. The magnitude of the c's indicates the maximum value of their respective subscripts permitted in any array element reference. The DIMENSION statement appears before the first executable statement of the program.

Multilevel dimensions are always written from left to right in increasing order of inclusiveness of the groupings in a table. In other words, minor to major. When referencing a table, the subscripts used follow the same order.

No reference to a subscripted variable is permissible unless the dimensions of the variable have been previously declared. No subscripted variable reference may contain a subscript which, during execution of the object program, may assume a value larger than the maximum specified for it in the DIMENSION statement. In a subprogram, no DIMENSION statement may contain the name of the

7.2. DIMENSION STATEMENT (Continued)

subprogram. No program making reference to a subprogram may contain a DIMENSION statement listing the name of the subprogram.

The name of an array without parentheses may also appear in a preceding type statement. If an array is dimensioned in a DIMENSION statement, it must not be dimensioned in a COMMON statement. Names used in a DIMENSION statement of an internal subprogram override global declarations of names including implied ones for that internal subprogram.

7.2.1. ADJUSTABLE DIMENSIONS

In an internal or external subprogram, an array name may appear in a DIMENSION statement. If this array name is a formal argument, the magnitude of the maximal subscript values may be indicated by unsubscripted INTEGER variables, provided the latter are also formal arguments. The number of dimensions and the maximal subscript values do not necessarily have to be the same as those of the actual argument which is an array in the calling program. The only restriction is that the number of locations determined by the DIMENSION statement in the subprogram must not exceed the number of locations provided by the DIMENSION statement in the calling program in which the name of the actual argument appears.

Examples:

1.
```
DIMENSION KARAY(10,20), LARAY(50,10,3)
```

2.
```
PARAMETER  J = 10, K = 20, L = 50
DIMENSION  KARAY(J,K),LARAY(L,J,3)
```

3.
```
      SUBROUTINE  ADDT(NA,NB,N)
      DIMENSION  NA(N),NB(N)
      DO  10  J = 1,N
   10 NA(N) = NA(N) + NB(N)
      RETURN
```

7.3. EQUIVALENCE STATEMENT

7.3.1. The EQUIVALENCE allows a storage unit to be referenced by more than one name. This statement is used to conserve storage. For instance, storage containing an array of data during one part of a program can contain variables during a succeeding part of the program which no longer requires the array.

The EQUIVALENCE statement contains groups of two or more names in the same program unit which are to share the same storage location. More than one group of names can appear in an EQUIVALENCE statement. The general form of an EQUIVALENCE is as follows:

EQUIVALENCE ($name_1$,$name_2$,....,$name_n$),($name_1$,$name_2$,....,$name_n$),....

A name can be a variable name, an array element name, or an array name.

The names which are to reference the same location must be separated by commas and enclosed in parentheses. Each parenthesized group of names is separated from succeeding groups of names by a comma.

Note that an EQUIVALENCE statement does not provide a mathematical definition; it allows a storage unit to be referenced by different names.

An example of an EQUIVALENCE statement containing an array and a list of variables, R,S, and T, is as follows:

```
DIMENSION SLUG(3)
EQUIVALENCE(SLUG(1),R),(SLUG(2),S),(SLUG(3),T)
```

Note that if one element of an array is equivalenced to one element of another array, all of the arrays are equivalenced.

For the purpose of the EQUIVALENCE statement, the appearance of the variable name without a subscript is taken to mean the first element in the array.

If a list contains three or more elements, or if the same variable name occurs in more than one list, the possibility of inconsistency arises. For example, the statement

```
EQUIVALENCE (A(3),C(2)),(A(4),D(3)),(C(2),D(3))
```

implies the equivalence of A(3),A(4). Inconsistencies of this nature are not permitted.

7.3.2. USES OF THE EQUIVALENCE STATEMENT

1. The EQUIVALENCE statement is primarily used to conserve storage space.

   Example: If two or more variables are used for temporary storage and their use is never overlapped, they may be put in an EQUIVALENCE statement so they will be assigned the same storage space.

2. It may be used to force the arrangement of storage of non-COMMON variables (both subscripted and non-subscripted variables). For instance, it may be desirable to have the variables R, T, B and A stored consecutively in storage; this can be done with

```
EQUIVALENCE (R,V(1)),(T,V(2)),(B,V(3)),(A,V(4))
```

   For simplicity assume that all variables involved are of the same type. In the same program, a READ Statement may then be given to read either R, T, B, or A, depending on a second quantity:

```
READ (Unit, F)K, V(K)

if K = 1 then V(K) is R
   K = 2 then V(K) is T
   K = 3 then V(K) is B
   K = 4 then V(K) is A
```

3. Formal arguments of internal or external functions or subroutines are considered to be independent of each other. If two actual arguments of a function reference or subroutine CALL are specified by the same variable, they are generally considered to be two different variables within the body of the subprogram. The appearance of formal arguments in an EQUIVALENCE statement of the subprogram will cause the compiler to use the location(s) of their corresponding actual arguments. For instance, if F(X,Y) is a function statement, X,Y and V are variables of the same type, and X and Y are assigned no values within the function body, the function reference F(V,V) will not cause the function body to have a common value for X and Y during execution. However, with X and Y appearing in an EQUIVALENCE statement, the location of V is used whenever X or Y are referenced.

4. It may be used to allow a storage area to be referenced using different names having different dimensions and/or different types.

```
DIMENSION NA (100), NB(1), B(100)
EQUIVALENCE (NA(1), NB(1), B(1))
```

7.3.2. USES OF THE EQUIVALENCE STATEMENT (Continued)

5. When a variable name has been inadvertently changed, an EQUIVALENCE statement can be used to equate the names to one location rather than correcting all the references to an intended variable.

7.4. COMMON STATEMENT

7.4.1. GENERAL

This statement permits program units to share work storage areas and has the general form:

$$\text{COMMON}/X_1/a_1,a_2, \ldots, a_n/X_2/a_{n+1},a_{n+2}, \ldots, a_m$$

in which X is any name of from one to six alphabetic or numeric characters, the first of which is alphabetic, and a is any variable or array name. X is called a block name, a name which bears no relationship to any variable or array name which may occur in the same or any other program unit. In any given COMMON statement, the variables occurring between block name X and the next block name or the end of the statement are in COMMON Block X. All variables from the beginning of the statement until the appearance of a block name, or all variables in the statement if no block name appears, are in "blank" or unlabeled COMMON. Alternatively, the appearance of two slashes with no block name between them assigns the variables which follow to "blank" COMMON.

A given COMMON Block name may occur more than once in a COMMON Statement or in a program. The compiler will string together in a given COMMON Block all variables so assigned, in the order of their appearance. The size of a COMMON Block is the sum of the storage required for each element introduced through COMMON and EQUIVALENCE Statements. The sizes of "blank" COMMON in the various programs and subprograms which may be run together need not be the same. The name of a formal argument must not appear in a COMMON statement.

7.4.2. COMMON BLOCK SIZE RULE

The sizes of all labeled COMMON Blocks of the same name must be the same in all programs which may be executed together.

Dimension and Common

All array names used in a program or subprogram must have their dimensions decalred in that program prior to their occurence in executable statements. An array name which is to be assigned to COMMON may have its dimension declared in a DIMENSION statement or simply by appending the dimension information to the array name in the Common Statement. It must not be dimensioned in both statements.

7.4.2. COMMON BLOCK SIZE RULE (Continued)

Dimension and Common (Continued)

For example:

```
COMMON X, Y (12,2,6), Z
```

would require a DIMENSION statement for X, if X were meant to be an array. However, the DIMENSION information for Y, in the example, has been given in the COMMON statement, and in this case must not occur in a DIMENSION statement. Names appearing in COMMON statements may also appear in preceding type statements.

Note, also that DOUBLE PRECISION and COMPLEX variables require special consideration, since they each occupy two computer words. A DOUBLE PRECISION variable is stored in two adjacent words with the most significant part in the first word. COMPLEX variables are, likewise, stored in two adjacent words with the real part occupying the first word. References to COMPLEX or DOUBLE PRECISION variables in EQUIVALENCE statements are always references to the first word of the corresponding word pair.

If, for instance, A is a COMPLEX array, B a REAL array and C a REAL unsubscripted variable, the statement

```
EQUIVALENCE (A(1), B(1)), (B(2), C)
```

would cause B(1) to occupy the real and C to occupy the imaginary part of A(1). But, under the same conditions the statement

```
EQUIVALENCE (A(1), B(1)), (A(2), B(2)),
```

is inconsistent. If, however, A were REAL, this statement would be consistent, but the second list would be redundant.

Note that if one element of an array is equivalenced to one element of another array, the entire arrays are equivalenced.

For purposes of planning the sharing of storage, the following statements show ways of storing new values in a location:

1. An arithmetic or logical statement will cause a value to be stored for the variable on the left-hand side of the equal symbol (=).

2. Executionof a DO will sometimes store a new value of the index variable.

3. Execution of a ASSIGN j to k will store a value in k.

4. Execution of input statements will store values in locations specified by the input list.

7.4.2. COMMON BLOCK SIZE RULE (Continued)

Dimension and Common (Continued)

5. Execution of a CALL statement may cause values to be stored in variables given as parameters or in common storage.

6. Reference to an external function may cause new values to be stored in common storage.

For purposes of arithmetic optimization, when a variable which appears in an EQUIVALENCE Statement has been defined by any of the above six items, then all equivalenced variables will be assumed defined.

The compiler will ignore equivalencing a formal argument to a non-argument variable. For equivalencing two formal arguments, see Section 7.3.2. USES OF THE EQUIVALENCE STATEMENT.

7.4.3. COMMON USED WITH EQUIVALENCE

When an array is named in an EQUIVALENCE statement and in a COMMON statement, the equivalence is established in the general way. This may increase the size of the COMMON block of storage and thus change the correspondences between the COMMON block described and some other COMMON block in another program.

For instance, consider these statements:

```
DIMENSION A(4), B(4)

COMMON  A,C

EQUIVALENCE (A(3), B(1))
```

Without the EQUIVALENCE statement, the COMMON block would contain five storage locations in the sequence A(1),A(2),A(3),A(4),C.

With the EQUIVALENCE statement the B array is brought into COMMON and requires the following sequence of storage location:

```
A(1),A(2),A(3),A(4),C

          B(1),B(2),B(3),B(4).
```

COMMON is now six storage locations long, since A(3) is equivalent to B(1),A(4) to B(2), and C to B(3). B(4) is the sixth variable of COMMON. Beware of contradictions. It is illogical to equivalence a variable in a COMMON block to another variable in the same block.

7.4.4. SOME USES OF COMMON

Ordinarily a variable within an external subprogram is not the same variable as that which was previously defined by the same name in the main program. However, if the programer wants it to mean the same thing, he can mention it in a COMMON statement in both the main program and the external subprogram. The two variables, although they are still distinct in principle, are assigned to the same storage location.

Another use of COMMON is to use it to assign locations to variables and to arrays in the order desired. The assignment of storage addresses is in consecutive order as the variables appear in the COMMON statement, or statements, with those in blank COMMON appearing first.

Example of COMMON:

In the following statement:

```
COMMON  A,B,C/ITEMS/X,Y,Z//D,E,F
```

the variables A,B,C,D,E and F will be placed in blank common in that order; the variables X,Y, and Z will be placed in the named common area ITEMS.

The same effect may be produced thus:

```
COMMON  A,B,C

COMMON  //D,E,F

COMMON  /ITEMS/X

COMMON  /ITEMS/Y

COMMON  /ITEMS/Z
```

7.4.4. SOME USES OF COMMON (Continued)

Example of COMMON: (Continued)

Another feature of blank common is that variables or arrays that appear in a calling program or a subprogram may be made to share the same storage locations with variables or arrays (having different names) in other subprograms. For example, if one program contains the statement:

```
COMMON TABLE
```

and a second program contains the statement

```
COMMON TREE
```

the variables TABLE and TREE refer to the same storage locations.

If the main program contains these statements

```
REAL A,B,C

COMMON A,B,C
```

and a subprogram contains the statements

```
REAL X,Y,Z

COMMON X,Y,Z
```

then A,B, and C are in the same storage locations as X,Y and Z.

7.5. DATA STATEMENT

The DATA statement provides the assignment of data at compilation time. It is not an executable statement at object time; however, new values may replace the initial values through executable statements at run time, thus destroying the original contents.

The DATA statement must be preceded by all DIMENSION, COMMON, and Type statements that relate to the variable names in the DATA statement lists.

The form of the DATA statement is:

```
DATA List/Literal List/, List/Literal List/, ....
```

where List is a series of one or more variables or array names, and Literal List is the data to be placed within the variable or array.

If an array name appears without subscripts, enough data must be present to fill the array. If it is desired to partially fill an array, then subscripts or implied DO-Loops must be used.

Example 1:

```
DIMENSION CARGO(10), WEIGHT(10)

DATA CARGO,WEIGHT/20*100.0/
```

Example 2:

```
DIMENSION CARGO(10),WEIGHT(10)
DATA (CARGO(I),WEIGHT(I),I = 1,10,2)/2*0.0,8*100.0/
DATA CARGO(10) / 1.0/
```

Note the asterisk in the above examples. It is possible to repeat literals without explicitly rewriting them by use of an asterisk and of an unsigned integer preceding the literal, specifying the number of times that the literal is to be repeated.

Note that data should be bounded by slashes.

7.5. DATA STATEMENT (Continued)

The types of data which may appear in a DATA statement are:

1. Integer constants
2. Real constants
3. Hollerith constants (preceded by nH)
4. Octal constants (preceded by Q)
5. Double Precision
6. Complex

Examples of DATA statements:

```
DATA DICK,BOB,DON /12H END PROGRAM       , 7.8/

DATA A/67.87/, B/54.72/, C/5.0/

DATA A,B,C  /67.87,54.72,5.0/

DATA R,S,T,U,V,W  /6*21.7/

DATA N(1) /10/

DATA DOT,X,BLANK /1H.,1HX,1HΔ/
```

Note in the above statement, if the number of characters of text is not the same as the number of characters in a storage location, the data is left adjusted and space filled on the right.

```
DATA N1,N2  /Q77,Q7700/
```

Note in the above statement, if the octal characters of text are not sufficient to fill the storage location, the data is right justified and ignore filled on the left.

Example of Table Headers:

```
                                          C.72
     DIMENSION  M(20,30)                   ↕
     DATA ((M(I,J),I=1,20),J=1,2)         /60H
1         THIS    IS      A               ,60H
2              HEADER                     ,60H
3         EXAMPLE                         ,60H
4                                           /
```

Note that a comma is used between the data items of the list. The number of commas separating data items may not exceed 199.

7.6. EXERCISES

7.6.1. Prepare a DIMENSION statement for an array NA to reserve 1Ø records of 2Ø words each.

7.6.2. Prepare a DIMENSION statement and a DATA statement for an array NLABL, the contents of which are:

```
6,Ø,Ø,24HTHIS IS A TAPE LABEL
```

7.6.3. Code a subroutine to clear an array of any size. Use parameters.

7.6. ANSWERS

7.6.1.
```
DIMENSION NA(20,10)
```

7.6.2.
```
DIMENSION NLABL(7)

DATA NLABL/6,0,0,24HTHIS IS A TAPE LABEL
```

7.6.3.
```
SUBROUTINE  CLEAR(NARAY,N)

DIMENSION NARAY(N)

DO 1Ø  J = 1,N

10   NARAY(J) = Ø

RETURN
```

8. CENSUS LIBRARY FUNCTIONS

CENSUS LIBRARY FUNCTIONS

The following are the CENSUS written library functions available for data manipulation, extractions, and conversions.

$LPM_n(x)$	FORTRAN access to schedule parameter cards
J_{jf}(NAME,n)	Justify, shift and extract
$IXTRC_j$(NAME,n)	Extract from an array and justify
LStt(NAME,n)	Shift a word
LSSU(NA,MASK,NB)	Selective substitute
LADD(NA,NB)	Logical addition (inclusive OR)
LSUB(NA,NB)	Logical subtraction (exclusive OR)
LAND(NA,NB)	Selective clear (logical AND)
LFX(NAME)	Fieldata to XS3 Conversion
LXF(NAME)	XS3 to Fieldata Conversion
LFB(NAME)	Fieldata to Binary Conversion
LXB(NAME)	XS3 to Binary Conversion
LBF(NBIN,NFILL)	Integer binary to Fieldata Conversion
LBX(NBIN,NFILL)	Integer binary to XS3 Conversion
MSR($HXXXX)	Systems references; XXXX may be: DATE, TIME, JRID, or MACH.

8.1. $LPM_n(x)$

This function gives the programer access to parameter cards which may be read in as schedule cards. A maximum of ten parameter cards may be used, containing up to eleven words each, producing a maximum parameter array of 110 words. The mode of the LPM_n function is always integer. The parameter cards consist of Fieldata characters.

General Form
$LPM_n(x)$ n = the parameter card number (0 through 9) x = the word number on a card

Example 1:

To set the 5th word from the first parameter card into N:

```
N = LPM0 (5)
```

Example 2:

To set the first word from the last parameter card into K:

```
K = LPM9 (1)
```

Example 3:

To move the entire 110 word parameter array into KARAY, the following sequence could be coded:

```
      DO    110  J = 1,110
100   KARAY (J) = LPM0(J)
```

Note that the definition of "$LPM_n(x)$ is continuous throughout the entire 110 words. In other words, the expression "LPM0(110)" and "LPM9(11)" are equivalent.

Please note that data may be moved from the parameter area, but no data may be moved into the parameter area.

Example 4:

To test the contents of a parameter word against a specific Hollerith constant, the following sequence could be coded:

```
      DATA NHOLL  /6HΔΔTEST/
      IF (LPM0 (1).EQ.NHOLL) GO TO 999
```

8.2. J_{jf} (NAME,n)

This sets the function expression equal to a value obtained by extracting the Nth field (half, third, or sixth of a word) from "NAME" and justifying the result (either to the right, left, arithmetically, or no justification.

General Form
J_{jf} (NAME,n)
j = type of justification = R (right justification) = L (left justification) = A (arithmetic justification) * = N (no justification)
f = type of field to extract = H (half word) = T (third word) = S (sixth word)
NAME = the name of the word from which the extraction is made
n = the position of the field within "NAME" to be extracted for half word extraction it may be a 1 or 2 for third word extraction it may be 1, 2, or 3 for sixth word extraction it may be 1,2,3,4,5 or 6

* Arithmetic justification means there is an extension of the sign of the extracted field. For example:

if NAME = 363740414243

JAT(NAME,1) is OCTAL: 000000003637

because the first bit of the first character 3 (binary 011) is a zero.

On the other hand

JAT (NAME,3) is OCTAL: 777777774243

because the first character is 4 (binary 100) and its leading bit is a 1. Arithmetic justification is not effective on sixth word extractions, therefore leading zeros will always be present.

8.2. J_{jf} (NAME,n) (Continued)

EXAMPLES

If "NAME" = 363740414243 (octal) then the values of the expressions below are as shown:

Expression	Octal Value
JRS (NAME,2)	000000000037
JLH (NAME,2)	414243000000
JNT (NAME,2)	000040410000
JAT (NAME,1)	000000003637
JAT (NAME,2)	777777774041
JLS (NAME,1)	360000000000
JAS (NAME,4)	000000000041
JNS (NAME,5)	000000004200
JAH (NAME,2)	777777414243
JRT (NAME,3)	000000004243

8.3. IXTRC$_j$ (NAME,n)

This sets the function expression equal to a value obtained by extracting the Nth six-bit character from an array and justifying the result (either to the right, left, or no justification).

General Form
IXTRC$_j$ (NAME,n) j = type of justification = R (right justification) = L (left justification) = N (no justification) NAME = name of the array n = the character position in the array "NAME" (the maximum for n would be a number which is 6 times the number of words in the array)

Example:

If the array "INCARD" contains an image of an 80 column card and "IN" is an 80 word array and the following statements are executed, then 80 words of "IN" will contain the 80 characters of "INCARD". The results (in this case) are left justified, zero fill to the right.

```
      DO   10  I = 1,80
10    IN(I) = IXTRCL (INCARD,I)
```

8.4. LStt (NAME,n)

This sets the function expression equal to a value obtained by shifting "NAME" as described below.

General Form
LStt (NAME,n) tt = type of shift = RC (right circularly "n" bit positions) RL (right logically "n" bit positions) * LC (left circularly "n" bit positions) * LL (left logically "n" bit positions) * RA (right arithmetically "n" bit positions) ** n = the number of bit positions to shift NAME
* A logical shift results in the loss of bits and the substitution of zero bits. ** An arithmetic shift results in the loss of bits and sign extension to the left as in JRT.

EXAMPLES

If "NAME" = 010203040506 (all values shown in octal)

then the values of the following expressions are as shown:

Expression	Octal value
LSRC (NAME,12)	050601020304
LSRL (NAME,18)	000000010203
LSLC (NAME,6)	020304050601
LSLL (NAME,24)	050600000000
LSRA (NAME,18)	000000010203
LSRA (LSLC(NAME,5),35)	777777777777
LSRC (JNS(NAME,2),6)	000002000000
LSRL (NAME,36)	000000000000

8.5. LSSU (NA,MASK,NB)

This sets the function expression equal to a value obtained by combining two words on the basis of a mask.

General Form
LSSU (NA,MASK,NB) where the "1" bits in "MASK" define the bit positions that will be taken from "NA", and "0" bits in "MASK" define the bit positions that will be taken from "NB".

Example

If NA = 010203040506

and MASK = 770000770077

and NB = 767574737271

then the value of the expression LSSU(NA,MASK,NB) is 017574047206

8.6. LADD(NA,NB)

This sets the function expression equal to a value obtained by a "logical addition," or an "inclusive OR" (i.e., AND/OR) operation.

General Form		
LADD(NA,NB)		
(Bit in NA)	(Bit in NB)	(Result)
0	0	0
0	1	1
1	0	1
1	1	1

Example

If NA = 252525252525

and NB = 000077630077

then the value of the expression

LADD(NA,NB) is 252577672577

8.7. LSUB(NA,NB)

This sets the function expression equal to a value obtained by a "logical subtraction," or an "exclusive OR" (i.e. EITHER/OR) operation.

General Form

LSUB(NA,NB)

(Bit in NA)	(Bit in NB)	(Result)
0	0	0
0	1	1
1	0	1
1	1	0

Example

If NA = 765432102345
and NB = 777777000000
then the value of the expression
LSUB(NA,NB) is 012345102345

8.8. LAND(NA,NB)

This sets the function expression equal to a value obtained by a "logical AND" operation.

General Form

LAND(NA,NB)

(Bit in NA)	(Bit in NB)	(Result)
0	0	0
0	1	0
1	0	0
1	1	1

Example

If NA = 010203040506
and NB = 777700770000
then the value of the expression
LAND(NA,NB) is 01020040000

8.9. LFX(NAME)

This function changes the Fieldata characters of a word to the corresponding XS3 codes. If no corresponding code exists, the character becomes an ignore.

General Form
LFX(NAME)

Example

NXS3 = LFX(NFD)

where NFD is the Fieldata word to be used as input to the conversion, and NXS3 is the converted result in XS3.

Fieldata	Character	XS3	Fieldata	Character	XS3
00	@	00	40	)	43
01	[	00	41	-	02
02	]	00	42	+	63
03	#	35	43	<	00
04	CR	00	44	=	76
05	↓	01	45	>	00
06	A	24	46	&	16
07	B	25	47	≠	55
10	C	26	50	*	56
11	D	27	51	(	17
12	E	30	52	%	75
13	F	31	53	:	62
14	G	32	54	?	57
15	H	33	55	!	00
16	I	34	56	,	21
17	J	44	57	\	00
20	K	45	60	0	03
21	L	46	61	1	04
22	M	47	62	2	05
23	N	50	63	3	06
24	O	51	64	4	07
25	P	52	65	5	10
26	Q	53	66	6	11
27	R	54	67	7	12
30	S	65	70	8	13
31	T	66	71	9	14
32	U	67	72	'	15
33	V	70	73	;	23
34	W	71	74	/	64
35	X	72	75	.	22
36	Y	73	76	□	00
37	Z	74	77	≠	60

8.10. LXF(NAME)

This function changes the XS3 characters of a word to the corresponding Fieldata codes. If no corresponding code exists, the character becomes an ignore or a space.

General Form
LXF(NAME)

Example

NXS3 = LXF(NAME)

where NXS3 is the XS3 word to be used as input to the conversion; and NFD is the converted result in Fieldata.

XS3	Character	Fieldata	XS3	Character	Fieldata
00	¿	05	40	tab	05
01	¡	05	41	"	05
02	-	41	42	|	05
03	0	60	43	)	40
04	1	61	44	J	17
05	2	62	45	K	20
06	3	63	46	L	21
07	4	64	47	M	22
10	5	65	50	N	23
11	6	66	51	O	24
12	7	67	52	P	25
13	8	70	53	Q	26
14	9	71	54	R	27
15	'	72	55	$	47
16	&	46	56	*	50
17	(	51	57	?	05
20	ƒ	05	60	≥	05
21	,	56	61	ß	05
22	.	75	62	:	53
23	;	73	63	+	42
24	A	06	64	/	74
25	B	07	65	S	30
26	C	10	66	T	31
27	D	11	67	U	32
30	E	12	70	V	33
31	F	13	71	W	34
32	G	14	72	X	35
33	H	15	73	Y	36
34	I	16	74	Z	37
35	#	03	75	%	52
36	¢	05	76	=	44
37	@	00	77	Not Used	05

8.11. LFB(NAME)

This function converts a Fieldata numerical quantity containing no decimal points to integer binary. The input data must be right justified in a word with any fill character to the left of the high-order numeric. If signed, the sign must be adjacent to the high-order numeric.

General Form
LFB(NAME)

Example

```
NBIN = LFB(NFD)
```

The following are legal inputs:

INPUT	INTEGER BINARY RESULT
ΔΔΔΔ64	000000000100
000100	000000000144
////-100	777777777633
ΔΔ$010	000000000012
+00012	000000000014
-00008	777777777767
CRΔ+10	000000000012
999ΔΔ1	000000000001
ΔΔΔΔΔΔ	000000000000
////-0	777777777777
ΔΔΔΔΔ-	777777777777
ΔΔΔΔΔ+	000000000000
******	000000000000
ABCDEF	000000000000
-099.1	000000000001

8.12. LXB (NAME)

This function converts an XS3 numerical quantity with the same restriction as imposed on the function LFB.

General Form
LXB (NAME)

Example

```
NBIN = LXB (NXS3)
```

8.13. LBF (NBIN, NFILL)

This function converts an integer binary number to a Fieldata word with fill characters inserted to the left of the converted numeric digits if the conversion produces less than six characters. If the converted number results in more than six numeric characters, the value becomes "******".

General Form
LBF (NBIN, NFILL) Where NBIN is the integer binary number to be converted and NFILL is a word of Fieldata fill characters

The following are examples of input and output:

NBIN	NFILL	CHARACTERS ON OUTPUT
000000000100	ΔΔΔΔΔΔ	ΔΔΔΔ64
777777777677	ΔΔΔΔΔΔ	ΔΔΔ-64
000000000000	ΔΔΔΔΔΔ	ΔΔΔΔΔ0
777777777777	ΔΔΔΔΔΔ	ΔΔΔΔ-0
000000000144	ΔΔΔΔ$Δ	Δ$Δ100
347536144072	ΔΔΔΔΔΔ	******
777777777633	iiiiii	ii-100
000000000005	ΔΔΔCRΔ	ΔΔCRΔ5
430241633705	iiiiii	******
000000000200	ΔΔΔΔΔ+	ΔΔ+128
000000000014	ΔΔTOT=	TOT=12

8.13. LBF (NBIN, NFILL) (Continued)

Example 1:

a) DATA NFILL/1HΔ/

NFD = LBF (NBIN, NFILL)

or

b) NFD = LBF (NBIN, 1HΔ)

Example 2:

DATA NFILL /0010101010101/

NXS3 = LBF (NBIN, NFILL)

Example 3:

a) DATA NFILL /0/

NFD = LBF (NBIN, NFILL)

or

b) NFD = LBF (NBIN, 0)

8.14. LBX (NBIN, NFILL)

This function converts an integer binary number to an XS3 word with fill characters inserted to the left of the converted numeric digits if the conversion produces less than six characters. If the converted number results in more than six numeric characters, the value becomes "******".

General Form
LBX (NBIN, NFILL) Where NBIN is the integer binary number to be converted and NFILL is a word of XS3 fill characters.

Example

NXS3 = LBX (NBIN, NFILL)

Examples for input and output are the same as those for LBF.

8.15. MSR (4Hxxxx)

This FORTRAN function allows a program to access "real time" information such as the date, time of day, job identity, and the machine identification code.

General Form

MSR (4Hxxxx)

Where xxxx requests the following:

a) DATE - The current date in Fieldata will be returned as the value in the format: MMDDYY.

b) TIME - The time of day in milliseconds in integer binary is returned. This is wall-clock time. It is not useful for calculating the program's run time.

c) JRID - The value returned will be the Job Request Identity of the program in Fieldata, left justified, space filled to the right. (The JRID is not necessarily the program name.)

d) MACH - A single right-justified Fieldata character with binary zero fill to the left will be supplied as the value which represents the machine on which the program is running:

"A" is the 1107A

"B" is the 1107B

"C" is the 1108C

"D" is the 1108D

Examples

1) NA = MSR (4HDATE)
2) NB = MSR (4HTIME)
3) NC = MSR (4HJRID)
4) ND = MSR (4HMACH)

9. CENSUS LIBRARY SUBROUTINES

CENSUS LIBRARY SUBROUTINES

The following are the CENSUS-written library subroutines available:

CHKDIV($n) Divide fault recovery

NOCHK Turn off divide fault recovery

TYPE (n, nH MESSAGE) Display message
or
TYPE (n, nH MESSAGE,j,REPLY) Wait for a reply to a message

LOAD (unit,n,nH MESSAGE,$) A message to load a tape unit

CHANGE (unit,n,nH MESSAGE, $) A message to change a tape unit

INSRTf (NA,NB,n) Insert into a word

INSERT (NA, NARRAY,n) Insert into an array

LDStt (NA,NB,n,NC,ND) Shift and store two words

MOVEf (NA,n,NB,m) Move a field from one array to another

LB2X (NBIN, NXS3, NFILL) Integer binary to 2-word XS3

LB2F (NBIN,NFD,NFILL) Integer binary to 2-word Fieldata

UNPK (INPUT,EDIT,CORE) Unpack fields into separate words

PACK (CORE,EDIT,OUTPUT) Pack characters into fields of an array

SORT (LFILE,LPAR) Sort an internal array of fixed-length items

EX (NFNCT,NUNIT,NSTAT,NBUFF,NOWDS,NN) direct access to drum and tapes

9.1. CHKDIV ($n)

General Form
CALL CHKDIV ($n) where "n" is any statement number to which the logical flow is to be transferred when a divide fault occurs.

All divide faults (characteristic underflow, characteristic overflow, and divide overflow) occurring in a FORTRAN program will automatically return a result of zero.

The execution of CALL CHKDIV ($999) will interrogate the divide fault indicators. If any divide fault has occurred, control will be transferred to the statement labeled 999. All divide fault indicators will be reset at this point. The programer has control and may take any action deemed necessary.

9.2. NOCHK

General Form
CALL NOCHK

The execution of this statement will "turn off" the divide recovery feature. Any divide fault will then result in an error termination and dump.

9.3. TYPE (n, nHMESSAGE)

To display a message on the console printer.

General Form
CALL TYPE (n, nHMESSAGE) where "n" is the number of characters to type and "MESSAGE" is any Hollerith message (in Fieldata) to be displayed on the console printer

Example 1:

```
DIMENSION NMES(3)
DATA NMES  /17HTHIS IS A MESSAGE/
.
.
.
CALL TYPE (17,NMES)
```

Example 2:

```
CALL TYPE (17,  17HTHIS IS A MESSAGE)
```

9.4. Type (n,nHMESSAGE,j,JREPLY)

To ask a question which is displayed on the console printer and to wait for a reply to be typed by the operator.

General Form
CALL TYPE (n,nHMESSAGE,j,JREPLY) where "n" is the number of characters to type and "MESSAGE" is any HOLLERITH message (in Fieldata) to be displayed on the console printer, and where "j" is the maximun number of characters to accept in reply and "JREPLY" is an array to which the left justified HOLLERITH characters of the reply will be stored with space fill.

Example:

```
CALL TYPE (21,21HTYPE INPUT REEL COUNT, 1, NCOUNT)
```

9.5. LOAD (u,n,nHMESSAGE, $999)

To display a message calling for the loading of a tape unit.

General Form
CALL LOAD (u,n,nHMESSAGE, $999) where "u" is a FORTRAN unit number on which the tape is to be loaded, "n" is the number of characters in the "MESSAGE," and $999 is a statement label to which the subroutine will exit if there is a "N" reply to the load message which precedes the programer's message.

Example:

```
CALL LOAD  (10,11, 11HMASTER FILE, $1000)
```

9.6. CHANGE (u,n,nHMESSAGE, $999)

To display a message calling for the changing of a tape unit.

General Form
CALL CHANGE (u,n,nHMESSAGE, $999) where "u" is a FORTRAN unit number where a tape is taken off and a new one is mounted, "n" is the number of characters in the "MESSAGE," and $999 is a statement label to which the the subroutine will exit if there is a "N" reply to the change message which precedes the programer's message.

Example:

```
CALL CHANGE (10,11, 11HINPUT FILEA, $1000)
```

This statement will produce the following on the console printer.

```
nn JRID   -CH 2/6 INPUT FILEA
```

where "2/6" is the absolute channel and unit assigned this run to FORTRAN unit number 10(AT). A "Y" reply by the operator will cause the statement following the CALL line to be executed. A "N" reply will cause statement 1000 to be executed.

9.7. INSRTf (NA,NB,n)

To set the right justified contents of "NA" into a specified field of "NB". "NA" and "NB" are whole words.

General Form

CALL INSRTf (NA,NB,n)

where "f" is the type of field and becomes a part of the subroutine name:

f = "H" (transfer a half-word field)

f = "T" (transfer a third-word field)

f = "S" (transfer a sixth-word field)

and "n" is the position of the field in "NB" into which the insertion will take place.

for half-word insertion: 1 or 2

for third-word insertion: 1,2, or 3

for sixth-word insertion: 1,2,3,4,5, or 6

If "n" assumes a value which is larger than logically possible for the type of field involved, the insertion will not occur.

Example

If "NA" = 010203040506

and "NB" = 717273747576

and the following statement is executed.

CALL INSRTT (NA,NB,3)

then the final contents of "NB" is 717273740506

9.8. INSERT (NA,NARAY,n)

To set the right justified character from "NA" into the specified position in the array "NARAY".

General Form
CALL INSERT (NA,NARAY,n) where "n" may assume values ranging from 1 to 6 times the dimension of "NARAY" (the total number of characters in the array)

Example:

If the 80-word array "NOUT" contains 80 right justified characters and the following statements are executed.

```
          DO   10   I = 1,80

   10     CALL INSERT (NOUT (I),NPRNT,I)
```

then the 80 right justified characters from the 80 words of "NOUT" will be packed into the first 80 character positions of "NPRNT".

9.9. LDStt (NA,NB,n,NC,ND)

To shift a two-word field composed of "NA" and "NB" and to move the result to "NC" and "ND" leaving "NA" and "NB" unchanged.

General Form
CALL LDStt (NA,NB,n,NC,ND) where tt is the type of shift and forms the last two characters of the subroutine name: tt = "RC" (shift right circularly "n" bit positions) tt = "RL" (shift right logically "n" bit positions) tt = "LC" (shift left circularly "n" bit positions) tt = "LL" (shift left logically "n" bit positions) tt = "RA" (shift right arithmetically "n" bit positions)

Example:

If "NA" = 010203040506

and "NB" = 071011121314

and the following statement is executed:

```
CALL LDSRL (NA,NB,18,NC,ND)
```

then "NC" = 000000010203
and "ND" = 040506071011

9.10. MOVEf (NA,n,NB,m)

To move a specified field from the array "NA" to a specified field in the array "NB".

General Form

CALL MOVEf (NA,n,NB,m)

where f is the type of field to be moved and forms the last character of the subroutine name.

f = "H" (move half-word fields)

f = "T" (move third-word fields)

f = "S" (move sixth-word fields)

and "n" is the position of the field within the array "NA" and "m" is the position of the field within the array "NB"

"n" and "m" may assume values which fall within the limits of the array specified.

Example 1:

To transfer 5 successive half-word fields from one array to another (starting at the first field in "NA" and at the seventh field (word 4, first half) in "NB", the following could be coded:

```
      DO  100    I = 1,5
100   CALL MOVEH  (NA, I,NB,I + 6)
```

Example 2:

To transfer 6 constants, stored as the right half of each word in a six-word array, into a three-word array, packing two constants to a word.

```
      DO  50  I = 1,6
50    CALL  MOVEH (NARRAY, I*2, NCON,I)
```

9.11. LB2X (NBIN,NXS3, NFILL)

To convert an integer binary number to a two-word XS3 numeric field.

General Form
CALL LB2X (NBIN,NXS3,NFILL) where NBIN is an integer binary number to be converted, NXS3 is the first word of a two-word field which will contain the converted results, and NFILL is a word of XS3 fill characters which will be inserted to the left of the converted numeric digits.

The following are some legal inputs:

NBIN	NFILL	Hollerith Result
000000000100	Δ Δ Δ Δ Δ Δ	ΔΔΔΔΔΔΔΔΔ64
777777777677	Δ Δ Δ Δ Δ Δ	ΔΔΔΔΔΔΔΔ-64
000000000000	Δ Δ Δ Δ Δ Δ	ΔΔΔΔΔΔΔΔΔΔ0
777777777777	Δ Δ Δ Δ Δ Δ	ΔΔΔΔΔΔΔΔΔ-0
377777777777	//////	/34359738367
400000000000	//////	-34359738367
000000003766	Δ00000	Δ02038

Example 1:

XS3 space fill

```
DATA   NFILL  /0010101010101/
CALL   LB2X (NBIN, NXS3,NFILL)
```

Example 2:

XS3 ignore fill

```
CALL  LB2X (NBIN,NXS3,0)
```

9.12. LB2F (NBIN,NFD,NFILL)

To convert an integer binary number to a two-word Fieldata numeric field.

General Form
CALL LB2F (NBIN,NFD,NFILL) where NBIN is an integer number to be converted, NFD is the first word of a two-word field which will contain the converted results, and NFILL is a word of Fieldata fill characters which will be inserted to the left of the converted numeric digits.

Legal inputs are the same as for LB2X.

Example

Fieldata space fill:

a)
```
DATA NFILL /1HΔ/
CALL LB2F(NBIN,NFD,NFILL)
```

or

b)
```
CALL LB2F (NBIN,NFD,1HΔ)
```

9.13. UNPK (INPUT,EDIT,CORE)

The purpose of this subroutine is to unpack fields from input records and to store each field into separate consecutive locations.

General Form

CALL UNPK (INPUT,EDIT,CORE)

where INPUT is an array containing the fields of data to be unpacked under control of EDIT format to the array CORE.

EDIT Description

The EDIT format controls the scanning of INPUT.

There is no control over placement of the unpacked fields into CORE. Storage into CORE is consecutive. CORE is always right justified with binary zero fill to the left. No conversion of any type is performed on the unpacked characters.

The least significant digit of each edit code in the array EDIT specifies the number of characters which will be unpacked from INPUT and placed into the next available location in CORE.

The least significant digit may range from 1 to 6.

The repeat factor attached to the left of the least significant digit describes the number of fields with the same size. Writing "35" is equivalent to writing "5,5,5". The repeat factor may range from 2 to 4096.

The negative edit code (a number preceded by -) resets the INPUT scan to a new word and character position. The least significant digit is the character position and this is preceded by the word location. This reset feature may move the scan either forward or backward in the INPUT record. There is no unpacking when this negative code is used. It merely moves the scanner.

The same EDIT code string, if applicable, may be used by UNPK and PACK.

During the first reference to UNPK, the edit codes are analyzed and modified by the subroutine. For this reason, the code string should not be changed by the programer.

The last edit code in EDIT must be a binary zero to terminate the action.

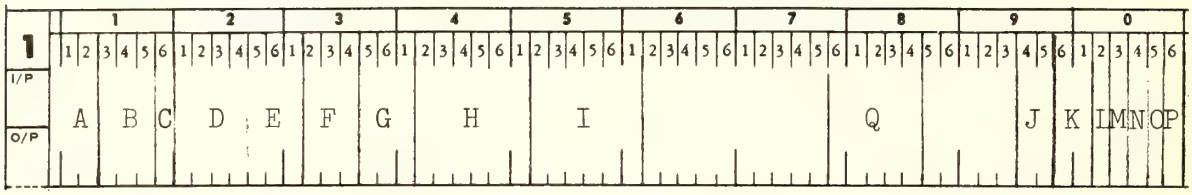
9.13. UNPK (INPUT,EDIT,CORE) (Continued)

Example:

To unpack 17 fields of the 10-word array INPUT, placing these fields into an array CORE, the following could be coded:

```
      INTEGER  INPUT(10), EDIT(12), CORE(17)
      DATA EDIT  /2,3,1,4,33,26,-94,22,51,-76,5,0/
      CALL UNPK  (INPUT,EDIT,CORE)
```

INPUT



As a result of the above edit codes, the following locations will contain the following fields picked up from INPUT.

	Field	No. of Digits Picked Up	
CORE (1)	A	2	
CORE (2)	B	3	
CORE (3)	C	1	
CORE (4)	D	4	
CORE (5)	E	3	
CORE (6)	F	3	
CORE (7)	G	3	
CORE (8)	H	6	
CORE (9)	I	6	
			Scanner reset to -94 (word 9, digit 4)
CORE (10)	J	2	
CORE (11)	K	2	
CORE (12)	L	1	
CORE (13)	M	1	
CORE (14)	N	1	
CORE (15)	O	1	
CORE (16)	P	1	
			Scanner reset to -76 (word 7, digit 6)
CORE (17)	Q	5	

9.14. PACK (CORE, EDIT, OUTPUT)

The purpose of this subroutine is to pack fields from consecutive locations into an output array specifying how many digits are to be placed in output and the position locations.

General Form

CALL PACK (CORE, EDIT, OUTPUT)

where CORE is an array containing data to be packed into an OUTPUT array under the control of the EDIT format. Data from CORE is expected to be right justified.

The least significant digit of each edit code describes the number of characters which will be packed from the next available location in CORE into the OUTPUT array. The least significant digit may range from 1 to 6.

The repeat factor attached to the left of a single digit code describes the number of times the edit code is to be applied. Writing "42" is equivalent to writing "2,2,2,2". The repeat factor may range from 2 to 4096.

The negative edit code resets the OUTPUT scan to a new word and character position. This reset feature may move the scan either forward or backward in the OUTPUT record.

Fields in the OUTPUT record which are not filled by the packing operation will have their previous contents intact.

No conversion of any type is performed on the packed characters.

The same EDIT code string, if applicable, may be used by UNPK and PACK.

During the first reference to PACK, the edit codes are analyzed and modified by the subroutine. For this reason, the code string should not be changed by the programer.

The last edit code in EDIT must be a binary zero to terminate the action.

9.15. SORT (LFILE, LPAR)

The purpose of this subroutine is to sort an internal array of fixed-length items into either an ascending or descending sequence. This is an unsigned binary sort (used primarily for sorting alphanumeric information). All information to be sorted must be contained internally in the program in a dimensioned array.

Prior to any reference to the sort subroutine, a DATA statement must be set up which will describe the array to be sorted. The DATA statement should appear as follows:

```
DATA (LPAR(I),I=1,n)/ K,ITMSIZ,NOITMS,NOKEYS,KP1,KM1,...../
```

where:

n	is the number of parameter entries in the DATA statement.
K	is equal to Ø for an ascending sort or equal to 1 for a descending sort.
ITMSIZ	is the number of words contained in one fixed-length item in the array to be sorted.
NOITMS	is the number of items in the array to be sorted.
NOKEYS	is the number of key fields in each item.
KP1	is a number which represents the position of the word which contains the first or most significant key field. (The first word of an item is word Ø)
KM1	is an octal mask which is used in extracting the key from the above word position in an item. Only the bit positions in the key word which correspond to "one" bits in the mask word are used as the actual key.
	The preceding two parameters (KP1 and KM1) may be repeated as required for the remaining keys in the order of decreasing significance.

The actual CALL to the sort subroutine appears as follows:

```
CALL SORT (LFILE,LPAR)
```

where LFILE is the array to be sorted and LPAR refers to the DATA statement containing the sort parameters.

9.15. SORT (LFILE, LPAR) (Continued)

EXAMPLE:

To sort an array of 4 one-word items into an ascending sequence using the third and first characters as the keys, the following could be coded:

```
DIMENSION  LPAR(8), LFILE(4)
DATA (LPAR(I), I=1,8) / Ø,1,4,2,Ø,0 ØØØØ77ØØØØØØ,Ø,0 77ØØØØØØØØØØ /

C**   ASSUME THE FOUR ITEMS CONTAIN THE FOLLOWING INFORMATION:
C**
C**   LFILE(1) CONTAINS THE SIX HOLLERITH CHARACTERS: CBADEF
C**   LFILE(2) CONTAINS THE SIX HOLLERITH CHARACTERS: CBBXYZ
C**   LFILE(3) CONTAINS THE SIX HOLLERITH CHARACTERS: BBAZZZ
C**   LFILE(4) CONTAINS THE SIX HOLLERITH CHARACTERS: DCAKLL

CALL SORT (LFILE,LPAR)

C**   THEN THE ITEMS WOULD BE ARRANGED AS FOLLOWS:
C**
C**   LFILE(1) CONTAINS THE SIX HOLLERITH CHARACTERS: BBAZZZ
C**   LFILE(2) CONTAINS THE SIX HOLLERITH CHARACTERS: CBADEF
C**   LFILE(3) CONTAINS THE SIX HOLLERITH CHARACTERS: DCAKLL
C**   LFILE(4) CONTAINS THE SIX HOLLERITH CHARACTERS: CBBXYZ
```

9.16. EX: A FORTRAN INPUT/OUTPUT SUBROUTINE

The purpose of this subroutine is to provide direct access to drum, IIA, or IIIA tapes. This subroutine is designed for special use and is intended in no way to replace the functions of FOLION or FORTRAN I/O. Specifically, this subroutine may be used for handling tapes which are in a format not compatible with the present system, or for using drum as a random access device. Each reference to this subroutine simply transfers information directly in and out of core from tapes or drum. It does not recognize any special format, nor does it provide any checksum protection.

The CALL line to perform an I/O function appears as follows:

```
CALL EX (NFNCT, NUNIT, NSTAT, NBUFF, NOWDS, NN)
```

NFNCT is a function code describing the type of I/O operation to be performed. The various I/O function codes are listed in the Exec manual (UP2577 Rev. 2 Appendix E).
Some of the more frequently used function codes are:

34_{10} = Read tape forward one block

02_{10} = Write tape forward one block

16_{10} = Rewind tape

17_{10} = Rewind tape with interlock

34_{10} = Read drum

02_{10} = Write drum

NUNIT is a Fortran unit number describing the IIA or IIIA tape unit, or the drum channel, on which the function is to be performed. These unit numbers are described in 10.2.6.

NSTAT is a variable into which the subroutine will set the status code of the completed I/O operation. This variable should be checked after every CALL to the EX subroutine. A complete list of these status codes can be found in the Exec manual (UP2577 Rev. 2 Appendix F). Some of the more frequently occurring status codes are:

Ø = I/O operation completed normally
3 = On a read tape forward operation, there was no more information recorded on the tape.
4 = When writing on a IIIA tape, the physical end of tape has been reached.

NBUFF is the name of a previously DIMENSIONed buffer area. This is the area from which information will be transferred to tape or drum by a write operation; or, this is the area into which information will be transferred from tape or drum by a read operation.

9.16. EX: A FORTRAN INPUT/OUTPUT SUBROUTINE (Continued)

NOWDS is the number of words that will be involved in the transfer of information in or out of core during the I/O operation.

1 - For a write tape function, NOWDS will be the number of words transferred from core to tape. This will also be the physical block size on the tape.

2 - For a read tape function, NOWDS will be the maximum number of words to transfer in from tape to core. If the block being read is larger than this maximum, then the excess words at the end of the block will be dropped -- they will not be transferred into core.

3 - For a write drum function, NOWDS will be the number of words transferred from core to drum.

4 - For a read drum function, NOWDS will be the number of words transferred from drum to core.

NN serves the following functions:

1 - For a write tape function, NN is not coded.

2 - For a read tape function, NN will be set by the subroutine with the actual number of words transferred from tape to core. For example, if NOWDS is equal to 200 and the block being read is actually 120 words in length, then 120 words will be transferred into core and NN will be set equal to 120 by the subroutine. If NOWDS is equal to 200 and the block being read is actually 250 words in length, then 200 words will be transferred into core, the last 50 words in the block will be dropped, and NN will be set equal to 200 by the subroutine.

3 - For drum reads or writes, NN specifies a relative location within the drum file where reading or writing will take place. The relative location of the first word in a drum file is Ø. When using drum, NN is not automatically incremented by the subroutine during a string of reads or writes. It is your responsibility to update this relative drum address prior to each CALL to the subroutine. For this reason, this subroutine is particularly useful in cases where it is advantageous to read information from a drum file in a random manner rather than reading straight through the file from front to back.

9.16. EX: A FORTRAN INPUT/OUTPUT SUBROUTINE (Continued)

Examples:

To read a 120 word block from IIA tape (Fortran Unit No. 1), the following could be coded:

```
      DIMENSION NBUFF (121)
      CALL EX (34, 1, NSTAT, NBUFF, 121, NN)
C**   CHECK STATUS OF COMPLETED READ:
      IF (NSTAT .EQ. 3) GO TO [END-OF-RECORDING ON TAPE]
      IF (NSTAT .NE. Ø) STOP ERROR
C**   CHECK BLOCK SIZE:
      IF (NN .NE. 120) STOP ERROR
```

To read a series of 20 word groups from drum (Fortran Unit No. 25) and write them out using FOLION the following could be coded:

```
      DIMENSION NBUFF (20)
      NN = Ø
      DO 5Ø J = 1, 1ØØ
      CALL EX (34,25, NSTAT, NBUFF, 2Ø, NN)
      IF (NSTAT .NE. Ø) STOP ERROR
      CALL ITMOUT (6HANYFIL, NBUFF)
50    NN = NN + 2Ø
```

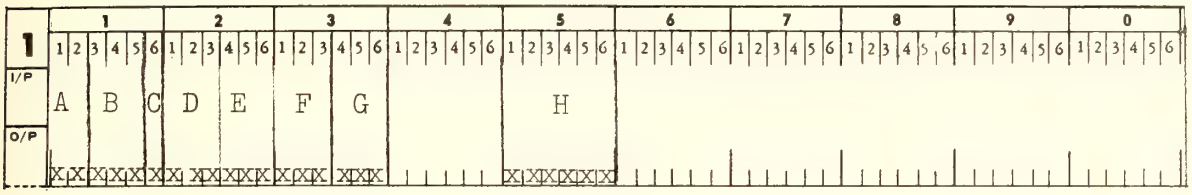
To rewind a IIIA tape (Fortran Unit No. 10) with interlock, the following could be coded:

```
      CALL EX (17, 10)
```

9.17. EXERCISES

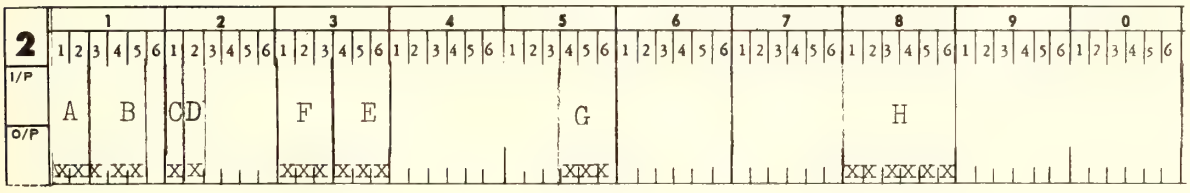
9.17.1. Prepare the edit codes in an array NEDIT (?) to unpack fields A, B, C, D, E, F, G, H of array INPUT (5) into an array NWS (8). Prepare a DIMENSION STATEMENT, a DATA STATEMENT, and a CALL to the UNPK subroutine.

INPUT



9.17.2. Prepare the edit codes in array NEDIT (?) to pack fields A, B, C, D, E, F, G, H of an array NWS (8) into an output array NOUT (8). Use the negative (-) indicator to reset the pointer to the specified output location. Prepare a DIMENSION STATEMENT, a DATA STATEMENT, and a CALL to the PACK subroutine.

NOUT



9.17. ANSWERS

9.17.1.
```
DIMENSION NEDIT (7), INPUT (5), NWS (8)
DATA NEDIT / 2, 3, 1, 43, -51, 6, Ø /
CALL UNPK (INPUT, NEDIT, NWS)
```

9.17.2.
```
DIMENSION NEDIT (13), NWS (8), NOUT
DATA NEDIT / 2, 3, -21, 21, -34, 3, -31, 3, -54, 3, -81, 6, Ø /
CALL PACK (NWS, NEDIT, NOUT)
```

10. FOLION

10.1. LION FORMATS

At the Bureau of the Census, LION subroutines are used for reading and writing records in either External or Internal formats. FOLION, a modification of LION, will read and write only in the Internal format. The FORTRAN READ and WRITE statements have been modified to use the LION External format. Internal and External formats are described below:

The Internal format is designed for recording data in the same form as the data appears in core store. This format is used for those files which are to be used for internal transfer of data within the computer system. The data need not be in absolute binary form. Refer to 19.19.

The External format is designed for recording data in Fieldata images for use with the punched card system or on-line High Speed Printer. This format is used for those files which are either input or output for the service or auxiliary routine which controls the operation of the on-line card and printer equipment.

10.2. FOLION

FOLION is a collection of subroutines which should be used by the programer in most instances for reading tapes and for creating tapes in LION internal format. The other methods for reading and writing are FORTRAN reads and and writes and the use of the EX subroutine. However, for general use, FOLION subroutines should be referenced. The advantages to be gained by using FOLION are many:

1. Labels may be recorded and checked.
2. Check sums and record counts are created and checked.
3. Records are collected into blocks when written and delivered one at a time when read.
4. Errors are displayed.
5. For 3A tapes, variable length records may be written or read.
6. Tapes may be read backwards.
7. End-of-reel procedures may be automatic.
8. Reels may be rewound with interlock.
9. Census compatible IIA tapes (recorded with 120 words per block) may be read or written.

10.2.1. INTRODUCTION

FOLION is linked to a FORTRAN coded program by the use of FORTRAN CALL statements. These CALL statements make reference to the following functions within LION:

1. Open file
2. Item handling
3. Close reel
4. Close file

10.2.1.1. FILE LABELS

Every file generated by the LION output subroutine will contain a physical label block. This block may or may not contain program file information. If some identification in the form of a file label is required, the location of the core image must be presented in the label argument of the file opening statement. This label image, if present, will be recorded at the beginning of each output file.

10.2.1.1.1. IIIA LABELS

The formation of the label image contains a date, reel number, and N words of label information. Thus, the label image itself consists of N+2 words. The first word of the label array must contain the total number of words in that label. Thus, the label array will have a length of N+3 words. For example:

```
DIMENSION LABEL(7)
DATA LABEL /6,Ø,01000000,21HTHIS IS A LABEL BLOCK/
```

In this example LION will supply the date and the reel numbering sequence starting with reel 1. However, if either the date or reel number is not equal to zero, LION will process the file with the specified date and/or reel number. The reel number will be incremented for each subsequent reel or file.

When a IIIA reel is written, ignores in the left half of word 3 of the label array will signal that the label characters which follow are to be considered as XS3 characters when this tape is used as input at a later date. When XS3 characters are required, use the octal equivalent in the DATA statement. If Fieldata characters are desired as the label then "01000000" should be placed into word 3 of the label array.

10.2.1.1.1. IIIA LABELS (Continued)

When checking labels using the automatic feature of FOLION, if it is desired to skip the check on selected words; then the array which contains the label to be compared with the input label may contain ignores for each word to be skipped in corresponding positions.

10.2.1.1.2. IIA LABELS

Since there is no provision for recording the date and reel number on IIA, the above example would appear as follows:

```
DIMENSION LABEL(5)
DATA LABEL /4,21HTHIS IS A LABEL BLOCK/
```

The four label words should be converted to XS3 before use.

10.2.2. OUTPUT SUBROUTINE

The output subroutine provides for writing tape or drum files with data originating in a FORTRAN object program in LION internal format.

10.2.2.1. OPEN OUTPUT FILE

The functions of the Open Output statement include the formation of a label block for each file as well as each reel.

Each Open Output statement must contain a unique Hollerith file name, which is used in associating the various references to the file. Any attempt to reopen an open file, write or close a closed file results in a File Control error code typed on the console typewriter.

The format of the OPen Output Statement is:

```
CALL OPNOUT (A, B, C, D, E, F, G, H, I)
```

where:

A = Hollerith File name (unique to this file)
B = File Data area (previously defined in a DIMENSION statement)
C = File Data area length (block size + 35)
For IIA, C = 155
For IIIA Fixed length items, C = 64Ø (recommended)
For IIIA Variable length items:
C = N(item length + 1) + 4Ø
where N = No. items per block.

10.2.2.1. OPEN OUTPUT FILE (Continued)

D = Item length for fixed length items. *
= Ø for variable length items. (See parameter C in ITMOUT).
E = Output device. This is an integer variable or constant. Its absolute value specifies the physical device to be used.
F = Alternate output device. Same as E.
= Ø if not used or when drum is used.
G = The location of the label to be written on the output reel(s).
= Ø for no label.
H = File control error return statement number. After the file control error is typed out, control will be returned to the FORTRAN program via this statement number.
= Ø for no exit return. After the file control error is typed, control will be returned to EXEC at $ERR.
I = End-of-reel return statement number. Control will be returned to the FORTRAN program via this statement number when the end of tape is detected.
= Ø or omitted for: 1) drum files, 2) and when LION is to perform an automatic end-of-reel procedure.

* ON IIA, the item size must be fixed length to any factor of 120.

10.2.2.2. WRITE OUTPUT FILE

There are no restrictions on the number of Write statements which may reference a single output file. Each Write statement must contain the unique Hollerith file name assigned to the output file.

The format of the Write statement is:

```
CALL ITMOUT (A, B, C)
```

where:

A = Hollerith File-name
B = Address of item to be written
C = Length of output item for variable length item
= Ø for fixed length items, or can be omitted. (See parameter D in OPNOUT.)

10.2.2.3. CLOSE OUTPUT REEL

10.2.2.3.1. The close Reel statement is used when the user wishes to terminate the writing of an output reel, and automatically initiate writing on the second or subsequent reels of the same file. Each Close statement must contain the unique Hollerith name assigned to the output file.

The format of the Close Reel statement is:

CALL OENREL (A,B)

where:

A = Hollerith File-name
B = Hollerith Tape positioning literal for terminated reel.

One of the following alphabetic literals should be used to indicate tape procedure:

6HREWIND: rewind without interlock
6HREWINT: rewind with interlock
6HNOREWI: no rewind

If one of the above literals is not present, the REWINT will be the resulting procedure.

10.2.2.3.2. The ITMOUT CALL which caused the sensing of the end of tape will not be performed until that reel has been closed, and the label block is written on the next reel. If the end-of-reel argument (I) of the CALL OPNOUT statement is equal to zero, the end-of-reel procedures will be automatically performed and control returned to the statement following the CALL ITMOUT. However, if the end-of-reel procedure is to be executed by the FORTRAN program, LION will exit to the end-of-reel statement number specified by argument I of the CALL OPNOUT statement. After the OENREL CALL has been performed, the item return statement must be executed to re-enter the program sequence beginning with the last ITMOUT CALL.

The format of the item return statement is:

CALL ITMRTN (A)

where:

A = File-name

10.2.2.4. CLOSE OUTPUT FILE

The Close file statement is used to terminate the writing of an output file.

The format of the Close file statement is:

CALL OENFIL (A, B)

where:

A = Hollerith File-name
B = Hollerith Tape positioning literal for the terminated file.

One of the following alphabetic literals should be used to indicate the tape procedure:

6HREWIND: rewind
6HREWINT: rewind with interlock
6HNOREWI: no rewind

If one of the above literals is not present, the REWINT will be the resulting procedure.

For Drum argument B could be Ø, or omitted.

10.2.3. INPUT SUBROUTINE

The Input subroutine provides for reading tape or drum files written in the LION internal format.

10.2.3.1. OPEN INPUT FILE

The functions of the Open Input statement include the reading and checking of the label block. If the label check fails, the tape label and its image is typed on the console printer with options for recovery. Each open input statement must contain a unique Hollerith file name, which is used in associating the various references to the file. Any attempt to reopen an open file, read or close a closed file results in a File Control error code typed on the console printer.

10.2.3.1. OPEN INPUT FILE (Continued)

The format for the Open Input statement is:

CALL OPNIN (A, B, C, D, E, F, G, H, I, J, K)

where:

A = File-name (unique to this file)
B = File data area (previously defined in a DIMENSION statement)
C = File data area length (block size + 35)
For IIA, C = 155
For IIIA Fixed length items, C = 64Ø (recommended-not required)
For IIIA Variable length items,
C = N(item length +1) + 40
where N = No. items per block
D = Input device. This is an integer variable or constant, its absolute value specifies the physical device to be used.
E = Alternate input device. Same as D.
= Ø if not used, or when drum is used.
F = The location of the label to be compared with the tape or drum label. Image words of 0 are not checked. *
= Ø if the label checking is to be omitted.
G = An integer variable or constant indicating tape direction.
= Ø for forward processing.
= 1 for backward processing.
H = File control error return statement number. After the file control error is typed out, control will be returned to the object program via this address. (If argument is ≠ to Ø.)

For IIIA Input:

I = End-of-reel return statement number. Control will be returned to the FORTRAN program via this statement number when the end of an input reel is detected.
= Ø when LION is to perform an automatic end-of-reel procedure.

* Whenever there is a label check failure, a message is typed out giving the channel and unit, file name, image to compare (30 characters), reel number, and tape label (30 characters). A reply of "Y" will effect an abnormal exit. If the answer is "N", the assumption is that a new reel has been mounted and a new label check initiated.

10.2.3.1. OPEN INPUT FILE (Continued)

For IIA Input:

I = Input item size. This value will be returned via the C parameter of the CALL ITMIN statement. *

The following two arguments are optional:

J = The location of the array to which the input file label will be transferred.

On IIIA, the input label will be transferred to the array as follows:

NO. OF LABEL WORDS, DATE, REEL NO., LABEL WORDS . . . .

On IIA, the input label will be transferred to the array as follows:

NO. OF LABEL WORDS,LABEL WORDS.....

K = Length of the array specified in the above argument (J).

Note that the array specified in the J argument of OPNIN may also be named in the G argument of OPNOUT and the input file label will be used as the output file label

* This value must be a factor of 120.

10.2.3.2. READ INPUT FILE

There are no restrictions on the number of Read statements which may reference a single input file. Each Read statement must contain the unique Hollerith file name assigned to the input file.

The instructions generated by the Read statement provide for filling the object program's request for an individual input item. The item is moved to a specified area, and the item length is conveyed via an integer variable.

The format of the Read statement is:

CALL ITMIN (A, B, C, D)

where:

A = Hollerith File-name
B = Location of area in which the requesting item is to be stored. This area must be a length as large as the maximum item size to be read.

10.2.3.2. READ INPUT FILE (Continued)

C = An integer variable into which the item length will be stored. For IIIA Multi-file input, if the end-of-file exit (below) is taken, C will assume one of the following values:

C = Ø, no more files follow on this tape

C < Ø, another file follows on this tape. In this case, to access the next file, the normal procedure would be to close input file with no rewind and then reopen input file.

D = End-of-file return statement number. When the end of file is detected, control will be returned to the program via this statement number.

An example of coding to process multi-file reels:

```
900     IF (NWDS) 902,   ,

        CALL  IENFIL ( file,6HREWINT)

        CALL CHANGE (u,n,nH MESSAGE, $999)

901     CALL OPNIN ( file, ----------)

        GO TO label to read next record

902     CALL IENFIL ( file, 6HNOREWI)

        GO TO 901

999     any ending procedure
```

10.2.3.3. CLOSE INPUT REEL

10.2.3.3.1. The close Reel statement is used when the user wishes to terminate the reading of an input reel, and automatically initiate reading on the second or subsequent reels of the same file. Each Close Reel statement must contain the unique Hollerith name assigned to the input file.

The format of the Close Reel Statement is:

```
CALL IENREL (A, B)
```

where:

A = Hollerith File-name
B = Hollerith Tape positioning literal for terminated reel.

One of the following alphabetic literals should be used to indicate tape procedure:

6HREWIND: rewind without interlock
6HREWINT: rewind with interlock
6HNOREWI: no rewind

If one of these literals is not present, the REWINT will be the resulting procedure.

10.2.3.3.2. The ITMIN CALL which detected the end of an input reel will not be performed until that reel has been closed and the next input label has been checked. If the end-of-reel argument (I) of the CALL OPNIN statement is equal to zero, the end-of-reel procedures will be automatically performed and control returned to the statement following the CALL ITMIN (A, B, C). However, if the end-of-reel procedure is to be executed by the FORTRAN program, LION will exit to the end-of-reel statement number specified by argument I of the CALL OPNIN statement. After the IENREL CALL has been performed, the item return statement must be executed to re-enter the program sequence beginning with the last ITMIN CALL.

The format of the item return statement is:

```
CALL ITMRTN (A)
```

where:

A = File-name

10.2.3.4. CLOSE INPUT FILE

The Close file statement is used to terminate the reading of an input file.

The format of the Close file statement is:

CALL IENFIL (A, B)

A = Hollerith File-name
B = Hollerith Tape positioning literal for terminated file. One of the following alphabetic literals should be used to indicate the tape procedure:

6HREWIND: rewind
6HREWINT: rewind with interlock
6HNOREWI: no rewind

If one of the above literals is not present, the REWINT will be the resulting procedure.

For drum this argument could be = Ø, or omitted.

10.2.3.5. NOTE ON IIIA END-OF-REEL EXIT

If it is desired to gain control at the end of every reel to determine whether to end the reading of data at this point or not, then the end-of-reel exit may be coded to go to a point where the following procedures should be coded. If record counts are desired when end-of-reel is encountered, digit 3 of word 26 of the buffer area should be loaded with octal 1 before executing IENFIL.

```
900     CALL INSERT (1,BUF(26),3)

        CALL IENFIL ( file)

        CALL CHANGE ( u,n,nHMESSAGE,$999)

        CALL OPNIN ( file,---------)

        GO TO label to read next record

999     any ending procedure
```

10.2.4. FILE CONTROL ERRORS

All file control errors will be displayed on the console typewriter in the format:

File-name FC ERR XXX.

where XXX is the value associated with the particular error. A complete list of file control errors is in the LION - General Reference Manual, UP-2581, Rev. 2.

After the typeout has been displayed LION will either terminate the run or return control to the object program. This action is determined by the open file statement.

The following LION file control errors have no recovery procedures available to the FORTRAN program. These errors when detected will result in termination via EXEC $ERR.

F.C. ERR CODE	DESCRIPTION
5	An attempt to perform a CALL ITMOUT statement has been made, the file closed or demanding a CALL OENREL operation.
6	An attempt to perform a CALL ITMIN statement with the file not yet opened, or when a CALL IENFIL or IENREL is the only legitimate operation.
7	An attempt has been made to execute the End-of-Reel procedure on a closed input file.
8	An attempt has been made to execute the End-of-Reel procedure on a closed output file.
118	The file data area being requested is occupied by a previously opened file.
119	The number of opened files exceeds 20.
120	When opening an output file one of the required arguments (A, B, C or E) was not present. For opening an input file (A, B, C, or D) was not present. For ITMIN (A, B, or C) is not present. For ITMOUT (A or B) is not present. For any other operations argument A is not present.
121	The peripheral equipment definition is not a legal value, (Peripheral definitions are defined in NTAB$).
122	In an item read or write, a close reel or file, or an ITMRTN statement, the file data area requested is not defined.

10.2.5. A CODED EXAMPLE

The following program will copy a single file IIIA tape recorded in LION internal format. The input label will be transferred to the output tape.

```
      DIMENSION JB1(640),JB2(640),ITM(20),LABEL(15)
      CALL OPNIN(4HFILA,JB1,640.10,0,0,0,0,0, LABEL, 15)
      CALL OPNOUT(3HOUT,JB2,640,20,11,0,LABEL,0,0)
100   CALL ITMIN(4HFILA,ITM,N,$999)
C**   PROCESS 20 WORD FIXED LENGTH ITEMS **
      CALL ITMOUT(3HOUT,ITM)
      GO TO 100
999   CALL IENFIL(4HFILA)
      CALL OENFIL(3HOUT)
      END
```

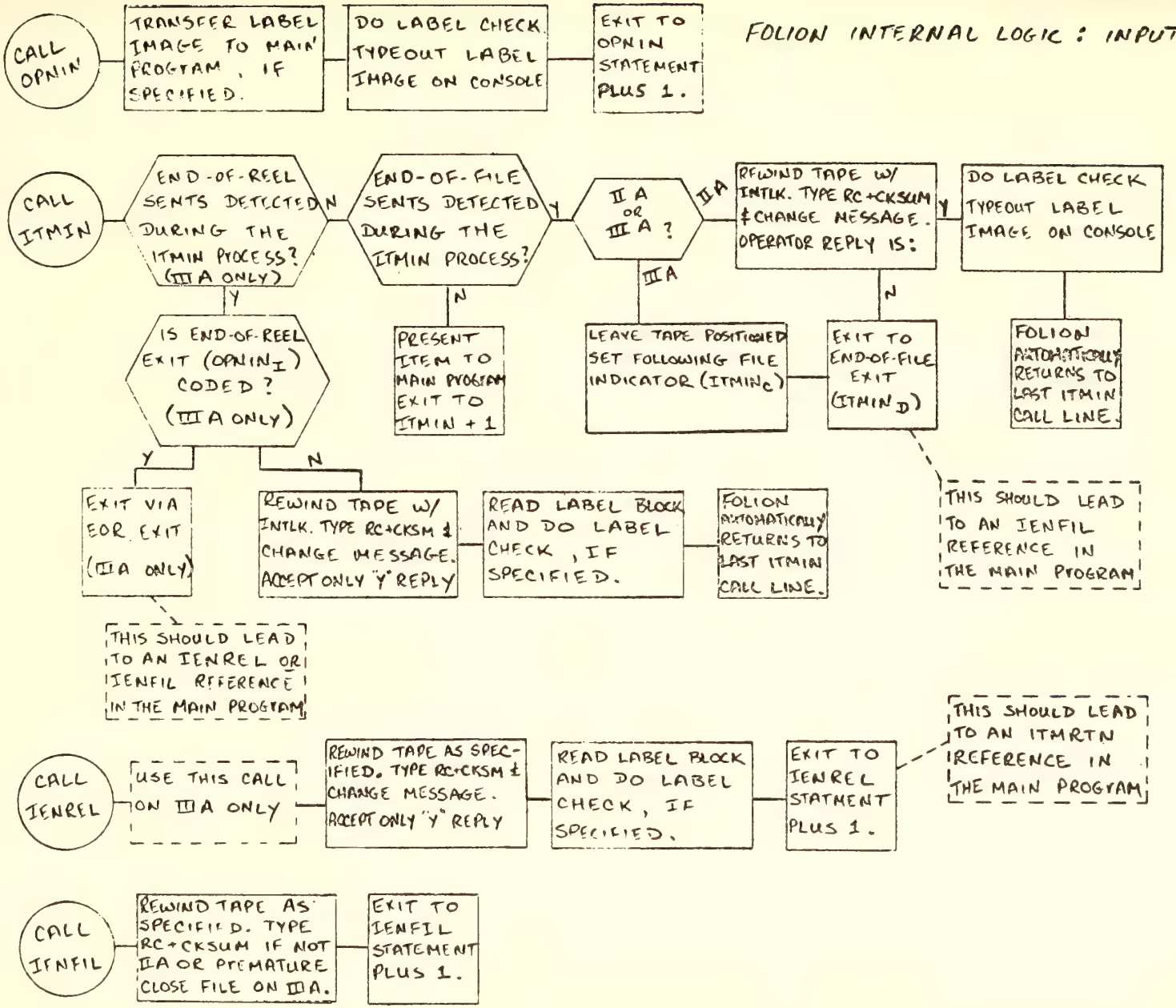
The facility (FAC) schedule card for the above program would appear as follows:

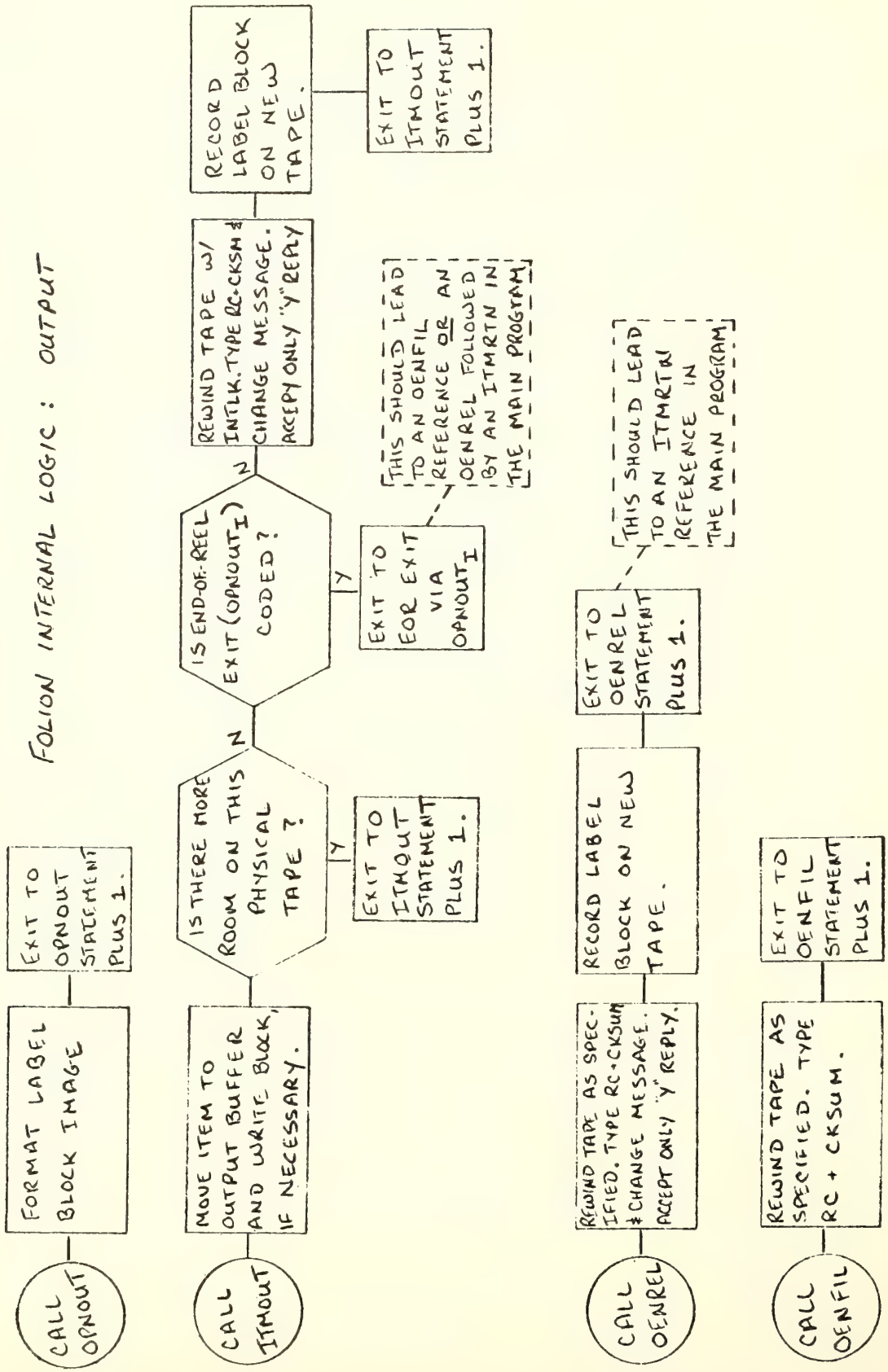
```
JRID,FAC,DB/2,IB/2,MDO/0,MTO/2:CT:DT:ET:FT:GT:PRNTT:PNCHT:READT.
```

10.2.6. FORTRAN I/O EQUIPMENT CONFIGURATION

UNIT NUMBER	EQUIPMENT DESCRIPTION	LOGICAL CHANNEL
- 3	Card Punch (SERVRO tape)	MTØ
- 2	Printer (SERVRO tape)	MTØ
- 1	Card Reader (SERVRO tape)	MTØ
+ Ø	Console Typewriter	
+ 1 through + 4	II-A tapes AA through DA	MAO
+ 5	Card Reader (SERVRO tape)	MTØ
+ 6	Printer (SERVRO tape)	MTØ
+ 7 through + 9	II-A tapes EA through GA	MAO
+10 through 16	III-A tapes AT through GT	MTO
+17 through 24	III-C tapes AC through GC	MCO
+25 through 29	Drum tables D1 through D5 with length tags LD1 through LD5	MDO

FOLION INTERNAL LOGIC : INPUT


CALL OPNIN
TRANSFER LABEL IMAGE TO MAIN PROGRAM, IF SPECIFIED.
DO LABEL CHECK TYPEOUT LABEL IMAGE ON CONSOLE
EXIT TO OPNIN STATEMENT PLUS 1.
CALL ITMIN
END-OF-REEL SENTS DETECTED DURING THE ITMIN PROCESS? (III A ONLY)
N
END-OF-FILE SENTS DETECTED DURING THE ITMIN PROCESS?
Y
II A OR III A ?
II A
REWIND TAPE W/ INTLK. TYPE RC+CKSUM & CHANGE MESSAGE. OPERATOR REPLY IS:
Y
DO LABEL CHECK TYPEOUT LABEL IMAGE ON CONSOLE
III A
N
Y
IS END-OF-REEL EXIT (OPNIN$_I$) CODED? (III A ONLY)
N
PRESENT ITEM TO MAIN PROGRAM EXIT TO ITMIN + 1
LEAVE TAPE POSITIONED SET FOLLOWING FILE INDICATOR (ITMIN$_C$)
EXIT TO END-OF-FILE EXIT (ITMIN$_D$)
FOLION AUTOMATICALLY RETURNS TO LAST ITMIN CALL LINE.
Y
N
EXIT VIA EOR EXIT (III A ONLY)
REWIND TAPE W/ INTLK. TYPE RC+CKSM & CHANGE MESSAGE. ACCEPT ONLY "Y" REPLY
READ LABEL BLOCK AND DO LABEL CHECK, IF SPECIFIED.
FOLION AUTOMATICALLY RETURNS TO LAST ITMIN CALL LINE.
THIS SHOULD LEAD TO AN IENFIL REFERENCE IN THE MAIN PROGRAM
THIS SHOULD LEAD TO AN IENREL OR IENFIL REFERENCE IN THE MAIN PROGRAM
CALL IENREL
USE THIS CALL ON III A ONLY
REWIND TAPE AS SPECIFIED. TYPE RC+CKSM & CHANGE MESSAGE. ACCEPT ONLY "Y" REPLY
READ LABEL BLOCK AND DO LABEL CHECK, IF SPECIFIED.
EXIT TO IENREL STATMENT PLUS 1.
THIS SHOULD LEAD TO AN ITMRTN REFERENCE IN THE MAIN PROGRAM
CALL IFNFIL
REWIND TAPE AS SPECIFIED. TYPE RC+CKSUM IF NOT II A OR PREMATURE CLOSE FILE ON III A.
EXIT TO IENFIL STATEMENT PLUS 1.


FOLION INTERNAL LOGIC : OUTPUT
CALL OPNOUT
FORMAT LABEL BLOCK IMAGE
EXIT TO OPNOUT STATEMENT PLUS 1.
CALL ITMOUT
MOVE ITEM TO OUTPUT BUFFER AND WRITE BLOCK, IF NECESSARY.
IS THERE MORE ROOM ON THIS PHYSICAL TAPE ?
N
Y
EXIT TO ITMOUT STATEMENT PLUS 1.
IS END-OF-REEL EXIT (OPNOUT_I) CODED ?
N
Y
EXIT TO EOR EXIT VIA OPNOUT_I
THIS SHOULD LEAD TO AN OENFIL REFERENCE OR AN OENREL FOLLOWED BY AN ITMRTN IN THE MAIN PROGRAM
REWIND TAPE W/ INTLK. TYPE RC-CKSM & CHANGE MESSAGE. ACCEPT ONLY "Y" REPLY
RECORD LABEL BLOCK ON NEW TAPE.
EXIT TO ITMOUT STATEMENT PLUS 1.
CALL OENREL
REWIND TAPE AS SPECIFIED. TYPE RC-CKSUM & CHANGE MESSAGE. ACCEPT ONLY "Y" REPLY.
RECORD LABEL BLOCK ON NEW TAPE.
EXIT TO OENREL STATEMENT PLUS 1.
THIS SHOULD LEAD TO AN ITMRTN REFERENCE IN THE MAIN PROGRAM
CALL OENFIL
REWIND TAPE AS SPECIFIED. TYPE RC + CKSUM.
EXIT TO OENFIL STATEMENT PLUS 1.

10.3. EXERCISES

10.3.1. To open a file using FOLION, a call must be made as follows: CALL OPNIN (A, B, C, D, E, F, G, H, I, J, K). Write the parameters required to open a file named 2HAT, using a buffer named BUF, and a label comparator named LABCK, no end-of-reel exit, and an array in which to store the new label.

10.3.2. Prepare the statement to read an item into a record array NWS from file 2HAT. Indicate an end-of-file exit and code for a change message.

ANSWERS

10.3.1. CALL OPNIN(2HAT,BUF,640,10,0,LABCK,0,0,0,LABEL,7)

10.3.2.

```
      20    CALL ITMIN(2HAT,NWS,N,$999)
            GO TO 100
     999    CALL IENFIL (2HAT)
            CALL CHANGE(10,8,8H ANY MORE,$9999)
            CALL OPNIN (2HAT,BUF,640,10,0,LABCK,0,0,0,LABEL,7)
            GO TO 20
    9999    .
            .
            .
            .
            .
```

11. GENERAL INFORMATION

March 1970

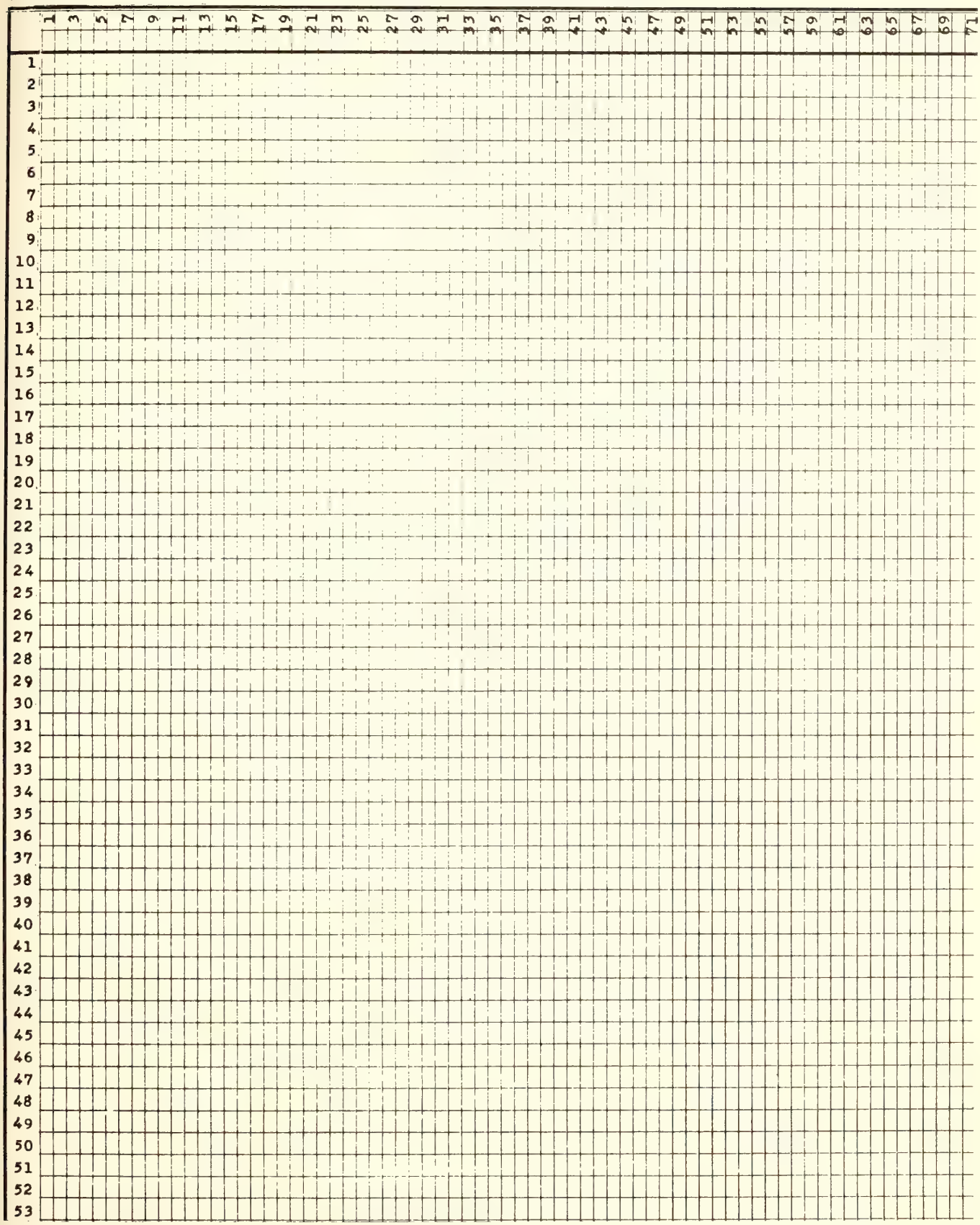
CENFOR 11-1

U.S. DEPARTMENT OF COMMERCE

BUREAU OF THE CENSUS

FORM BC-344 (FORMERLY 60-11-24)
(8-6-62)

HSP OUTPUT TABLE LAYOUT FORM



March 1970

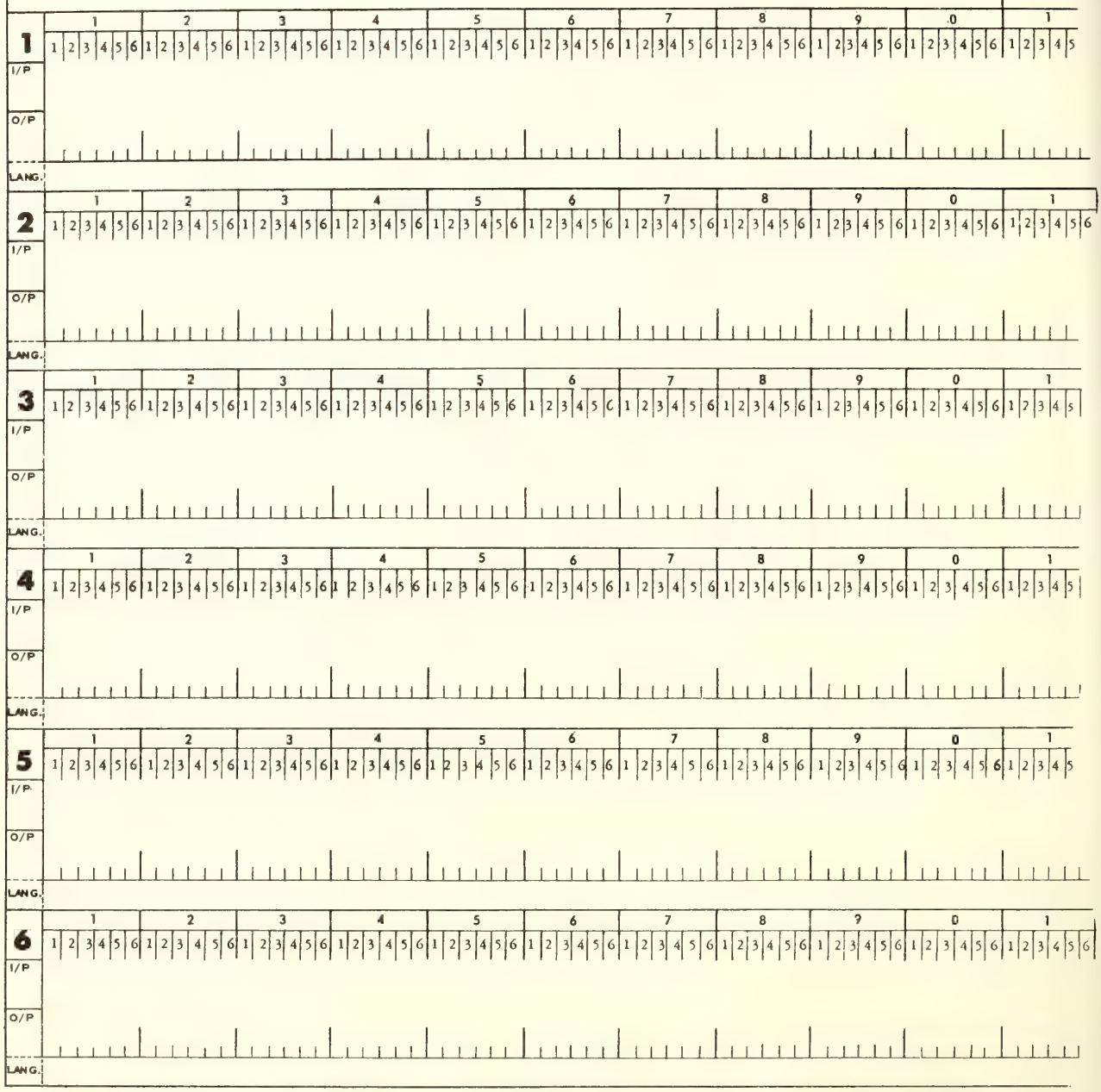
FORM BC-1508B
(3-3-67)

U.S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

PROGRAM

RUN

RECORD LAYOUT (FORTRAN)



11.1. PLANNING A HSP OUTPUT TABLE

Form BC 344 "HSP OUTPUT TABLE LAYOUT FORM" is available for planning the spacing of a table. Each column is numbered across the top of the sheet and each line is numbered down the sheet. It is good planning to prepare a dummy table, with all the required headings or spanners and stubs indicating the exact positions where arithmetic figures are to be placed before starting the coding of a program. This table becomes part of the documentation of the program.

11.2. DATA RECORD LAYOUTS

Form BC 1508B "RECORD LAYOUT(FORTRAN)" is available for displaying the locations of fields of data within a record. Usually this "layout" is supplied as a part of the job specifications. Each field may be given a name or a number. The written specifications in conjunction with the "layout" inform the programer where fields are located, what these fields contain, the size of a record, and the language of the data. They are a very essential part of the documentation of a program.

11.3. CREATING TEST DECKS

Rather than testing programs using the large amounts of data usually found in production, it is wise to test with a test deck, a smaller set of data which incorporates most of the possibilities to be found in actual production data.

Three general methods are used to secure or to create test data.

1. The first method is to request a copy of selected production data through your coordinator.

2. The second method is to write a program which generates output tailor-made to your needs.

3. The third method is to prepare information on a coding sheet in the form desired and to have cards punched and taped to your requirements. There are three ways to request this service depending upon the card volume desired:

Amount	Submit to	Remarks
1 - 100 lines	Special Services Branch	
100 - 500 lines	Input Preparation Branch	
500 lines or more	Input Preparation Branch	Must be scheduled through scheduler for your work area.

Information as to sentinels, fill characters, number of cards, tapes, taping program to be used, and machine to be used and any other pertinent information should be provided.

11.4. TAPE BLANKING AND STORAGE

While testing, it is the responsibility of the programer to periodically blank those test reels which he no longer needs. This releases the blanks for other uses; it also makes it easier to locate current test data. When programs become obsolete, their names should be furnished to Special Services for disposition.

When a program is released to PRODUCTION, directions for tape blanking and storage should be included in the operating instructions prepared for each program or survey. Special procedures have been established for permanent storage of vital data and for temporary use of this data whenever needed. Channel request through your survey coordinator.

11.5. SCHEDULE CARDS

Before a program can be selected as a candidate for loading and initiation, Schedule Cards must be submitted to the Executive System. The Schedule Cards consist of a series of input parameters to EXEC specifying the program to be initiated, its location, modifications to be made while loading, and all other pertinent information concerning the program. The schedule cards are read by EXEC from the specified input device, checked for legality, and stored in the schedule of Job Requests.

Every job has a minimum of three schedule cards. These are the priority card (PTY), the facility description card (FAC), and the last parameter card (PM.). Certain other cards may be present. These include the table length assignment card (TAL), the transfer card (TRN), other parameter cards (PMN), and the subroutine card (SUB). Their presence is optional and they are used if specific table lengths are to be increased, if facilities are to be transferred to succeeding jobs, if the job program requires starting parameters which are unique to each run, and/or subroutines are to be deleted or renamed at load time.

Special Services provides the schedule cards for an assembly, but the programer must furnish schedule specifications for his own programs.

11.5.1. PRIORITY CARD (PTY)

The PTY card is the basic card for every job request.

FIELD	COLUMNS	CONTENT
CARD COUNT	12-13	Total number of schedule cards written in decimal notation.
PROGRAM NAME	17-22	Name of job as found on START card, left justified.
MEDIUM	32	This field defines the type of input medium on which the job program resides. (a) "T" denotes UNISERVO IIIA (b) "A" denotes UNISERVO IIA (c) "C" denotes UNISERVO IVC (IIIC) (d) The symbol "*" denotes that the MEDIUM NAME field which follows contains an absolute assignment.
MIX	60	This field assigns the job to either the compute-limited class or to the I/O limited class of programs. (a) "P" denotes an I/O limited program. (b) "C" denotes a compute-limited program.
*TIME	62-65	Up to four decimal digits giving a high estimate, to the nearest minute, of the central computer time needed for the operation of the program.
LIERARY NAME	69-77	This field contains the name of the library tape. "Z" denotes that the program library is on the same tape as the program.

Any other fields needed, such as MEDIUM NAME, will be supplied by operations. A period will terminate the fields on each schedule card.

* See next page for additional information.....

Note: IIIC tape units are referred to as IVC tape units. These units are IBM compatible. Only 2 units are available per machine. The units in our system can record at 800 bits per inch. They are referred to as IVC tape units or uniservos.

11.5.1. PRIORITY CARD (PTY) (Continued)

*TIME (Continued)

OPTIONS FOR AUTOMATIC TERMINATION BASED UPON ESTIMATED RUN TIME

EXEC will recognize the first (high order) digit in the Estimated Time field on the PTY card as a termination control parameter and accept the other three digits as the estimated run time. The parameter is effective only when the field contains four decimal digits. If the field contains three or fewer decimal digits, the action is an effective zero option.

The three possible uses of a parameter in Estimated Time are as follows:

1. Ønnn (nnn=three decimal digits≤511)
   For each accumulation of nnn minutes of central processor time by the program, EXEC types "*JRID *EXCEEDED RN TIM⟨S⟩". There is no automatic termination.

2. 1nnn (nnn≤511)
   At the first recognition that the program has accumulated nnn minutes of central processor time, EXEC terminates the program using E65 as the code defining termination by reason of exceeding Estimated Time.

3. 2nnn (nnn≤255)
   At the recognition that the program has accumulated for the first time nnn minutes of central processor time, EXEC types "*JRID *EXCEEDED RN TIM ⟨S⟩". At the recognition that the program has accumulated for a second time nnn minutes of central processor time, EXEC terminates the program using E65 as the code defining termination by reason of exceeding Estimated Time.

Automatic termination parameters 1nnn and 2nnn will be applied to programs now released for production only upon request from the sponsor.

11.5.2. FACILITY DESCRIPTION CARD (FAC)

This card must be present for all jobs which are to be executed under EXEC control. Columns 12-77 contain three specifications, separated by commas.

(a) Amount of core needed: Approximate the number of words of core required by the program (and the library subroutines used) for instructions. Divide this number by 2048 (rounding the quotient up to the nearest whole number) to obtain the number of IBANK required. Likewise, divide the total number of locations needed for data (by the program and the library subroutines used) by 2048 (again rounding up) to get the number of blocks or banks of DBANK. In either case, add an extra block or two when in doubt. After the program has been assembled, the exact IBANK and DBANK values will appear on the ELF ALLOCATION printout. List the bank with the greatest number of blocks first; if they are equal, list IB first. After the first good assembly, submit correct schedule cards.

(b) Tape requirements: At least one tape unit must be used* for reading the input, and one is needed to store the output. There are two types of tape (channels) generally available, each with several units:

	III A			II A	
UNIT	TAPE	CHANNEL	UNIT	TAPE	CHANNEL
5 or -1	READT	MTØ	5 or -1	READA	MAØ
6 or -2	PRNTT	MTØ	6 or -2	PRNTA	MAØ
-3	PNCHT	MTØ	-3	PNCHA	MAØ
10	AT	MTØ	1	AA	MAØ
11	BT	MTØ	2	BA	MAØ
12	CT	MTØ	3	CA	MAØ
13	DT	MTØ	4	DA	MAØ
14	ET	MTØ	7	EA	MAØ
15	FT	MTØ	8	FA	MAØ
16	GT	MTØ	9	GA	MAØ

Example 1:

To schedule 2A units AA and BA (internal unit numbers 1 and 2), deleting all other units on the 2A channel, the following FAC card(s) should be coded:

```
JRID,FAC,DB/n,IB/n,MDØ/n,MAØ/2:CA:DA:EA:
JRID,FAC,FA:GA:READA:PRNTA:PNCHA.
```

*Some programs do not require tape inputs, e.g. a test deck generator.

11.5.2. FACILITY DESCRIPTION CARD (FAC) (Continued)

Example 2:

To schedule 2A units AA and BA (units 1 and 2) plus 3A unit AT (internal unit number 10) plus the 3A PRNTT tape (internal unit number 6 or -2), deleting all other units on the 2A and 3A channels, the following FAC card(s) should be coded:

JRID,FAC,DB/n,IB/n,MDØ/n,MAØ/2: CA:DA:EA:
JRID,FAC,FA:GA:READA:PRNTA:PNCHA,MTØ/2:BT:CT:DT:ET:
JRID,FAC,FT:GT:READT:PNCHT.

Example 3:

To schedule 3A units AT through DT (internal unit number 10 through 13), plus the 3A READT unit (internal unit number +5 or -1), deleting all other units on the 3A channel, the following FAC card(s) should be coded:

JRID,FAC,DB/n,IB/n,MDØ/n,MTØ/5:ET:FT:GT:

JRID,FAC,PRNTT:PNCHT.

Use of IIIC tapes is restricted.* See 10.2.6 for unit numbers. When writing tape facilities, specify how many are needed for each channel and delete ALL OTHERS for that channel by noting them, separated by colons, after noting the number used. List the channel with the greatest number of tapes first.

For example:

,,FAC,MTØ/3:BT:CT:DT:ET:FT:PRNTT:GT

,,FAC,MAØ/2:CA:DA:EA:FA:GA:PRNTA:

READA:PNCHA

(c) Drum requirements: If drum tables are being used, write:

MDØ/sum of all drum tables
and fill out a TAL Card.

If no drum locations are being used, write:

MDØ/Ø

A period follows the last entry of the FAC card.

*There is no provision for IBM formatted tapes. If a IIIC (or IVC) tape is specified, the format is expected to be LION internal if FOLION is used. When the subroutine EX is used, whole blocks are read or written.

11.5.3. TABLE LENGTH ASSIGNMENT CARD (TAL)

This card specifies increments to the minimum data table lengths which were established by the assembler. The card makes it possible to change at load time the amount of core or drum working area available to the program for a given run. Its inclusion is optional. The TAL card for drum is as follows:

,,TAL,LD1/n_1,LD2/n_2,.....,LD5/n_5.

UNIT	TABLE	LENGTH TAGS
25	D1	LD1
26	D2	LD2
27	D3	LD3
28	D4	LD4
29	D5	LD5

Named COMMON blocks may be extended by preceding the name of the block with L as:

,,TAL,LNAME/length of increase

11.5.4. TRANSFER CARD (TRN)

This card is included for a program which uses facilities which are to be transferred to the environment of a following program or which are to be received from other programs.

11.5.5. SUBROUTINE CARD (SUB)

An EXEC ROC program can be loaded with the subroutines which are specified as load-time subroutines in the source coding, renamed or deleted. The card format is as follows:

,SUB,a_1/b_1,a_2/b_2,...a_n/b_n

where

a_1,a_2....,a_n are the names given to the load-time subroutines or entry point at load-time.

b_1,b_2....,b_n are the new names to be given to the corresponding subroutines at entry point.

ROC means relative object coding.

11.5.6. PARAMETER CARDS (PMn)

A job program may require a set of input parameters to determine or select options of execution. These parameters are entered via the PMn Card, where n is a decimal digit from Ø to 9. On the PM. card it is necessary to write the division number and full project number beginning in column 12. For example:

,PM.,51ØØ83Ø1799Ø

Note: For fixed parameters, only one specification sheet is needed. For variable parameters, two specification sheets are required; part one for all other information for scheduling and part two with all parameter information. Also, there are separate forms for testing and production. These are color-coded for identification. The form numbers for testing are BC-1522 a,b,c. Those for production are BC-1521 a,b,c. For fixed parameters, the specification sheet is submitted only once unless changes become necessary. When variable parameters are used, a new part two specification sheet must be submitted with every request. When a program is released for PRODUCTION, duplicate specification sheets must be submitted.

Example:

Program name: EXAMPL

IB: 5ØØØ words

DB: 9ØØØ words

Card reader input: READT
Printer output: PRNTT
3 other IIIA tapes used: AT,BT,CT
1 IIA tape used: AA
2 drum tables used: LD1/1ØØØØ
LD2/5ØØØ
Total drum used: 15ØØØ
Estimated run time: 5 minutes

FORM BC-1522a
(4-6-67)

TEST
1107 EXECUTIVE SCHEDULE CARDS
FIXED PARAMETERS ONLY

U. S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

NOTE: Do not use this form for programs requiring more than one set of parameter cards. Use forms BC-1522b and BC-1522c

PROCESSING DIVISION USE ONLY

A. Division Code	B. Project Number	C. Project name	D. Programmer	E. Office telephone ext.
53	00833000	0 TRAINING	JOE BLOW	9999

F. Name of program (If general purpose service routine such as sort, enter suggested JRID.)	G. Job identification (work unit, group, State, period, etc.)	H. Request status	I. Date of request	J. Time of request
EXAMPLE	TRAINING	[X] Original [ ] Replacement	1/1/80	

Do not make entries in shaded columns. For all other columns, indicate the exact characters to be punched. Contents must be in accordance with EXECutive specifications.

JRID	TYPE	CC	PROGRAM NAME	RN	CDW	MEDIUM NAME	TKT	PTY	MX	TIME	MT	LIB. NAME
	PTY	05	EXAMPL		T		P	P		5	T	B

JRID	TYPE	FACILITY DESCRIPTIONS
	FAC	DB/5,IB/3,MT0/5:DT:ET:FT:GT:PNCHT,
	FAC	MA0/1:BA:CA:DA:EA:FA:GA:PRNTA:PNCHA:READA,MD0/15000.

JRID	TYPE	TABLE LENGTH INCREMENTS
	TAL	LD1/10000,LD2/5000

JRID	TYPE	RC	TRANSFERS
	TRN		
	TRN		

JRID	TYPE	OCFP & OCLP NAME
	SUB	

JRID	TYPE	PARAMETERS
	PM0	
	PM1	
	PM2	
	PM3	
	PM4	
	PM5	
	PM6	
	PM7	
	PM8	
	PM9	
	PM0	5300833000000

USCOMM-DC

March 1970

CENTOR 11-11

CENFOR 11-12 March 1970

FORM BC-559 (9-17-68) Ø U.S. DEPARTMENT OF COMMERCE BUREAU OF THE CENSUS

REQUEST FOR 1107 PROGRAM PREPARATION SERVICES

Section I – TO BE COMPLETED BY SPONSOR | PROCESSING DIV.

A. Div. code	B. Project number	C. Program name	D. Project name	E. Job number
53	ØØ833Ø ØØØØ	EXAMPL	TRAINING	

F. Requested by:				G. Required by: *(If urgent)*		H. Date/time submitted *(Stamp)*
1. NAME	2. TEL. EXT.	3. RM. NO.	4. BLDG. NO.	1. DATE	2. TIME	
JOE BLOW	9999	1600	3			

I. Services requested *(Check, unless otherwise indicated. Specify in "Remarks")*

1. PREPARE

a. CARDS	b. 1401 C/T	(1) FILL	(2) SENTINELS	c. OTHER *(Specify)*
☑ ASSY.	☐ NO	☐ IGNORE	☐ STANDARD	☐
☐ REASSY.	☐ YES	☐ ZERO	☐ NONE	
☐ OTHER *(Specify)*	*(Specify output tape label)*	☐ OTHER *(Specify)*	☐ OTHER *(Specify)*	

2. ASSEMBLE FORTRAN

a. TYPE	b. CORRECTIONS	c. LIBRARY
☑ ORIGINAL	☑ NO	☐ NO ☑ YES
☐ REASSEMBLY* FRAME WORK	☐ YES	☑ STANDARD ☐ OTHER *(Specify)*
☐ UTT ☑ RUTT		

*Unless otherwise specified, latest CSC will be used.

3. TEST

a.	b. IS MIDAS USED?	c. SCHEDULE CARDS	d. SEQUENCE *(If any)*	e. CROSS REFERENCE (PROCESSING)
☐ NO ☑ YES* EST'D. CENTRAL COMPUTER TIME 2 MINUTES	☑ NO ☐ YES	☑ SPECS ATTACHED ☑ ORIGINAL ☐ MODIFIED ☐ SAME AS LAST TEST	TEST NO. ____ OF ____	

*Do not test if assembly warnings exceed ____

4. TRANSCRIPTION

a. TYPE	b. REQUIRED	c. RECORD INFORMATION	d. NUMBER OF TAPES *(If known)*
☐ IIA TO IIIA	☐ PRIOR TO TEST ☐ TRANSCRIBE ONLY*	☐ FIXED RECORD SIZE ____ ☐ VARIABLE	(1) IN ____ (2) OUT ____ *(Specify any pertinent file information)*
☑ IIIA TO IIA	☐ PRIOR TO TEST ☑ AFTER TEST ☐ TRANSCRIBE ONLY*	☑ FIXED ☐ VARIABLE I/O DISPOSITION HSP	(1) IN 1 (2) OUT 1 *(Specify any pertinent file information)*

*Attach request for 1107 tape transcription.

5. LIBRARY MAINTENANCE

a. OWN CODING	b. EXTERNAL NAME OF LIBRARY	c. SCHEDULE CARDS	d. CROSS REFERENCE (PROCESSING)
☐ OCFP ☐ OCLP ☐ OTHER *(Specify)*		☐ SPECS. ATTACHED ☐ ORIGINAL ☐ MODIFIED ☐ CARDS ATTACHED ☐ SAME AS LAST REQUEST	

6. OTHER *Specify:*

J. Material submitted

1. ☑ CODING *(Number of lines)*	2. ☐ PUNCHED CARDS	3. OTHER *(Specify)*
673	a. APPROX. NO. ____ b. USE *(Specify)*	

K. Remarks

11.6. SUBMISSION OF PROGRAMS

11.6.1. Once a program has been completely coded, information indicating your specific instructions must be referred to the Special Services Branch in order that they will know the steps through which to take your program. It is always necessary to attach a "Request for 1107 Program Preparation Services" to the front of your coding sheets or your punched cards. Sections A-G must be filled out accurately with the requested information. Under Section I, the services required by your program should be circled in the left-most column. Information under each requested service should be specified. Under PREPARE AND ASSEMBLE check (√) whether it is an original or a reassembly. Under the label ASSEMBLE print "FORTRAN" (or "FORELF/ELF" for a sort) to indicate the language and note if there is to be a RUTT or an UTT. Check (√) if there are corrections, if it is a reassembly, and the type of library used. Check (√) YES if a test is desired, and NO under MIDAS because MIDAS is not available to FORTRAN programers. Note whether the schedule card specifications are attached to the coding or they are the same as they were for the previous assembly. If RUTT and/or UTT is requested, the details must be given under TRANSCRIPTION. If a library subroutine is needed the name must be specified under LIBRARY MAINTENANCE. The status of the schedule cards must be indicated for the subroutine also, and if your OWN CODING is present with a sort, this must also be noted. The amount of coding and/or the number of cards or any additional material submitted should be indicated. If more than 75 lines of coding are submitted, a cutoff time is not assigned until the punched cards are returned to Special Services. If fewer cards are to be punched a cutoff time will be assigned immediately and the punching will be done by Special Services. Any additional information which might be helpful to Special Services may be included under "REMARKS."

At the top of the page above the "R" in REQUEST a submission code must be supplied according to the following code list:

CODES FOR COMPUTER SUBMISSION REQUESTS

Code	Use
Systems Division Errors	
0	Original Assembly This category must be used for the first assembly of each new program. Any assemblies for name changes only should be coded as "not specified." "0" should also be used anytime to identify straight sorts, sorts with option coding, and merges.
1	Reassembly or Original Assembly Resulting From Specification Changes This category includes all changes in written or oral instructions given to the programer after the original assembly has been made.

11.6. SUBMISSION OF PROGRAMS (Continued)

11.6.1. (Continued)

Code	Use
Systems Division Errors (Continued)	
2	Logic Error Refers to programer logic only.
3	Coding Error Covers mistakes in instruction formatting or schedule card specifications and illegible writing.
4	Incomplete Test With No Reassembly Required Includes all testing not covered above which is required to prove in the program (new test data, incorrect estimation of run time, different parameter or switch options, etc.). This category excludes incomplete tests resulting from a Processing Division error (see Category A).
5	Production Run on Test Time
u	Not Specified Use for any condition which cannot be specified above. Do not use without supervisor's approval.
Processing Division Errors	
6	Schedule Card Error
7	Set-Up Error
8	Punch Error
9	Incomplete Test (instructions not followed completely).
Miscellaneous Errors	
A	Recreate Missing Data/Program Tapes (including HSP tapes).
B	Tape Read/Write Error Includes bad tapes or other hardware malfunction
R	Resubmission Used by Special Services only for requests incorrectly processed and caught prior to return to programer

11.6. SUBMISSION OF PROGRAMS (Continued)

11.6.1. (Continued)

Code	Use
Miscellaneous Errors (Continued)	
C	Other than computer service, such as punch test deck, print on the HSP, etc.
D	Service Run on Test Time

If the program is to be assembled only, this is the only sheet required. It, together with the coding, may then be submitted in a designated room where you must complete this form by stamping the date and time in the upper right-hand box. All listings will be returned to your office.

When a FORTRAN program is submitted for a reassembly, the programer should require that the previous PNCHT program tape be blanked. PNCHT tapes will be held only three days in case there should be a need for the old program.

FORM BC-558 (3-17-67)

U.S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

1107 OPERATING INSTRUCTIONS

A. Div. code	B. Project number	C. Program name	D. Sponsor's name	E. Tel. Ext.	F. Job number (PD)
53	ØØ833Ø ØØØØ	EXAMPL	TRAINING	9999	

G. Program information

1. [✓] TEST *(Other than sort/merge)*

2. TRANSCRIPTION
   - [ ] PRIOR TO TEST
     - [ ] UTT
     - [✓] RUTT
   - [✓] AFTER TEST

   APPROXIMATE NUMBER OF OUTPUTS: 3

3. [ ] SORT [ ] MERGE
   a. OPTION CODING USED: [ ] OWN [ ] DUMMY [ ] NONE
   b. EXTERNAL LIBRARY NAME
   c. PROGRAM TO BE USED *(If other than standard)*

4. LIBRARY MAINTENANCE
   a. TYPE OF SUBROUTINE: [ ] LOAD TIME [ ] ASSEMBLY TIME
   b. NO. OF COPIES REQUIRED
   c. EXTERNAL LIBRARY NAME

TO BE COMPLETED BY PROCESSING DIVISION

a. BINARY TAPE NO.	a. OWN CODING TAPE NO.	a. OWN CODING TAPE NO.
b. SOURCE [ ] FURNISHED [ ] FROM ASSEMBLY	b. SOURCE [ ] FURNISHED [ ] FROM LIBRARY MAINTENANCE	b. SOURCE [ ] FURNISHED [ ] FROM ASSEMBLY

H. Magnetic tape input(s)

[ ] NONE [✓] INDICATE

(If servo alternation specify in "M")

TAPE NUMBER (a)	TRANS. TAPE NO. (DPSD) (b)	ASSIGNED FACILITY NAME (c)	EXTERNAL LABEL (d)	REMARKS (e)
HM73Ø1		AT	DATATP	TRAINING SHELF
				HOLD Also RUTT

I. Magnetic tape output(s)

[ ] NONE [✓] INDICATE

(If servo alternation specify in "M")

ASSIGNED FACILITY NAME (a)	EXTERNAL LABEL (b)	TAPE NUMBER (DPSD) (c)	TRANS. TAPE NO. (DPSD) (d)	REMARKS (e)
BT	EXAMON			IIIA RUTT TO
CT	EXAMTW			IIA

L. Printing instructions

1105/1107 HSP						1107 HSP
BOARD	LOOP	PAPER	NO. COPIES	TAPE IDENT.	NO. BLOCKS TO PRINT	PROGRAM TO BE USED
3	3	143	1	AT	[✓] 100 [ ] ____ [ ] ALL	
3	3	143	1	BT	[✓] 100 [ ] ____ [ ] ALL	
3	3	143	1	CT	[✓] 100 [ ] ____ [ ] ALL	
					[ ] 100 [ ] ____ [ ] ALL	
					[ ] 100 [ ] ____ [ ] ALL	

M. Special instructions

USCOMM-DC

11.6.2. If a test is also to be made, the schedule card coding and and "1107 Operating Instructions" sheet must also be included. Sections A-G are self-explanatory.

11.6.2.1. MAGNETIC TAPE INPUTS

Enter the tape number. If the tape is a previously used tape, the number and external name may be obtained from other persons who have used it. If a request for a CENSER CT is submitted, you will not know the number which will be assigned to it, but you should refer to the external label and note that a CT will be done before the test. The next time that data is used, refer to the number which was assigned it.

Sometimes a series of programs will be tested. In this event the output(s) from one run may become the input(s) to another run. This should be noted wherever applicable in order to furnish the proper inputs to succeeding runs. Indicate the assigned facility name as the name which was not deleted on the schedule cards. If a production tape is to be used a copy of it must first be made.

11.6.2.2. MAGNETIC TAPE OUTPUTS

Similarly, with the output tapes, the assigned facility name should be indicated as with the input tapes, and the external label is whatever you wish to call it, usually the program name. The exact disposition of all output tapes should be specified. If the tape is to be transcribed, indicate. If the tape is to be held, indicate HOLD. If the tape is to be printed, indicate. If the tape is to become input to another run, indicate.

11.6.2.3. JUMP SWITCHES

There will be no switches in FORTRAN programs.

11.6.2.4. TYPEOUTS

Note only abnormal ones. These typeouts generally require action to either continue or discontinue running of your program and for this reason are most important.

11.6.2.5. PRINTING INSTRUCTIONS

Indicate the board and loop desired according to the "Summary of Characteristics of Standard High-Speed Printer Boards and Loops" in Chapter 19. The most commonly used set is:

Board 3, loop 3

The tape identification is the same name which was referenced under H and I "assigned facility name." Finally, indicate the number of blocks to be printed.

11.6.2.5. PRINTING INSTRUCTIONS (Continued)

A maximum of 100 blocks should be printed under test. If more data must be printed, indicate PRODUCTION PRINTING.

When you are through testing a program, Special Services should be told either to blank those tapes dealing with that program, or to put them in temporary or permanent storage.

11.6.3. When the program is ready to be released for production an "1107 Production Program Release" form must be submitted. Sections I and II of the form must be completely filled out and given, with duplicate, to your program coordinator with particular attention being given to the following:

A. This form is not required for a SORTN program.

B. Note Sorts and Merges with OCFP and/or OCLP in Section II, Item C.

C. Under Section II, omit Item F if Item E is "Initial Release."

D. Note carefully "type of program" (Section II, Item I) since they go on different libraries.

E. Consider the right box of Section II, Item J to mean prior production, not test specifications.

F. Use the left-hand box under Section II to include the following:

(1) Note if program is coded in FORTRAN

(2) External references for a program must be noted.

FORM BC-756
(10-7-68)

U.S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

1107 PRODUCTION PROGRAM RELEASE

TO BE COMPLETED BY PROGRAMMER

Section I AUTHORIZATION

A. Effective date and time
This form authorizes use of the program herein specified for production use after .....
Date | Time

B. Submitted on
Date | Time

C. Authorized by
Programmer's signature | Extension

Section II PROGRAM IDENTIFICATION

A. Div. code | **B.** Project No. | **C.** Name of program *(Must be same as PRO line)*

D. Last assembly
Date | Time

E. History
☐ Initial release
☐ Replacement for previous production version

F. Disposition of previous production version
☐ Blank
☐ To DATA storage

G. Future use
☐ Retain test version
☐ Testing completed

H. Suggested JRID
_ _ _ _ _ _ _ _ _ _ _

I. Type of program
☐ Production program
☐ Sort/Merge load time library

J. Specifications for production schedule cards
☐ Attached
☐ Test specifications reviewed and approved for production *(Date)* _____

TO BE COMPLETED BY PROCESSING DIVISION

Section III CONTROL

Special Services Branch

A. Type
☐ Single
☐ Stack
Group I.D. _____

B. Tape numbers
CSC
BIN
LTL

C. Librarian parameter card
Prepared by
Checked by

Section IV OPERATIONS

Operations Branch

A. Output tape number
CSC-1 _____ BIN-1 _____ LTL-1 _____
CSC-2 _____ BIN-2 _____ LTL-2 _____

B. Outputs verified
Controller's signature

Section V VERIFICATION

Special Services Branch

A. Processing verified and materials checked in by *(Signature of S.S. Branch Representative)*

Section VI TRANSFER TO PRODUCTION

Production Branch

A. Received by *(Signature of Production Branch Representative)*

USCOMM-DC

11.6.4. Also, when the program is ready for release to production, the "1107 Operating Instructions - Production" sheet must be completed and given to the program coordinator.

Items 1 through 6: Self-explanatory.

INPUT - Item 7: Must be completed, especially Item 7B. This is needed for identification as this name comes out on the teleprinter and Edit 75 log. This is the file name used in the program.

OUTPUT - Item 8: Must be completely filled out.

JUMP SWITCHES - Item 9: As they apply. None in FORTRAN.

NORMAL TYPEOUTS - Item 10: A normal typeout is defined as any information generated by you in your program for the necessary processing and control information that you specifically place in your run to aid in the overall operation of this program. If typeouts are needed that apply to sponsor information only, this must be shown; otherwise, the operators might become confused.

Example: "X MISSING ESTABLISHMENTS - NO ACTION
NECESSARY - SPONSOR INFORMATION ONLY."

ABNORMAL TYPEOUTS - Item 11: These typeouts generally require action to either continue or discontinue running of your program and for this reason are most important.

Note: Never end an "expected" or possible abnormal termination. Instead, program for a normal termination with a typeout such as "suspend further processing - save all outputs."

SPECIAL NOTES - Item 13: This category should include any information that you feel is particularly necessary above and beyond conditions set forth in your typeouts.

CONTROL INFORMATION - Item 14: Although this item is self-explanatory, either one of two types of control may pertain. If "input record counts equal output record counts," so note and no other control information should be necessary. If control is other than that information just stated, you should describe fully what type of control is necessary on your program so the control clerk, reading your instructions, can fully implement the standards you wish placed on your run.

FACILITY REQUIREMENTS - Item 15: Self-explanatory.

BLANKING PROCEDURES - Item 16: The disposition of magnetic tapes including blanking procedures should accompany the Operating Instruction(s) prior to the time a program is released for production.

PRINTOUTS TO - Item 17: Self-explanatory.

REFER QUESTIONS TO - Item 18: Self-explanatory.

FORM BC-1366 (9-19-66) U.S. DEPARTMENT OF COMMERCE BUREAU OF THE CENSUS **1107 OPERATING INSTRUCTIONS – PRODUCTION**	**1. Program title**	2. Date	3. Project No. *(If blank, refer to staging sheet)*
4. Originating division **Systems Division** **Processing Coordination Branch**	5. General title of series		6. BURR title or JRID

7. INPUT

Assigned facility name (a)	Control file tag (b)	External label (c)	Type of tape (d)	From (e)

8. OUTPUT

Assigned facility name (a)	Control file tag (b)	External label (c)	Type of tape (d)	To (e)

CENFOR 11-22

MARCH 1970

9. Jump switches	
10. Normal typeouts	Action
11. Abnormal typeouts	Action
12. Abnormal conditions	
13. Special notes	

USCOMM-DC

14. Control information

15. FACILITY REQUIREMENTS	I Bank	D Bank	Drum	III-A Servos	II-A Servos	IV-C Servos

16. Blanking procedures

17. PRINTOUTS TO:			18. REFER QUESTIONS TO:		
Name	Room No.	Extension	Name of programmer or PCB	Room No.	Extension

11.6.4. (CONTINUED)

It is the responsibility of the programer to prepare this information as completely as possible, for submission to the Processing Coordination Branch. If this program is one of a series, information for the entire series should be submitted together so that this knowledge and the instructions can be forwarded to the Processing Division as a complete "package." This package should include a copy of flowcharts, operating instructions and other general information deemed necessary for the computer processing of your program(s).

TIMING: On a one-time program, this information should be submitted to the program coordinator a minimum of 24 hours (one full work day) ahead of the anticipated running. If a series of programs is involved or a major segment of a census, this information should be submitted as early as possible and not later than seven to ten days prior to anticipated running.

In turn, these procedures will be reviewed for completeness, typed in final form and distributed from the Processing Coordination Branch. If you require review of these procedures prior to release, it is your responsibility to see that the information is supplied in ample time ahead of the time specified above.

11.7. PROGRAMER RESPONSIBILITIES

1. Generate test decks, or secure test data, that will test all paths of a program and hold testing time to a minimum.

2. Code, test, and debug the program. Use enough comments in the program to aid in later review or modifications.

3. Prepare initial draft of operating instructions for supervisor.

4. Note all modifications to the initial specifications in duplicate -- one copy to be filed with the program, one copy for the project leader.

5. Prepare necessary header strips.

6. Prepare a written explanation of all unique code systems used.

7. Upon completion of the program, provide documentation of the program, including:

   A. Name of program and project number.

   B. Program sponsor.

   C. Name of programer and supervisor.

   D. Purpose of the program (short narrative).

   E. Relationship with project (following "Program X" or before "Program Y").

   F. Computer system (programing language and machine orientation).

   G. Specifications from which program was designed:

      1. Input and output record layouts (where pertinent).
      2. Flow Charts, logic tables, and narrative specifications.
      3. Input and output file characteristics.
      4. Program restrictions and/or limitations.

   H. References to other frameworks or routines (e.g., "Option Coding Last Pass to Standard 1107 Sort Routine").

   I. Operating Instructions.

   J. Program printout and sample output listings.

   K. Other internal documentation deemed appropriate. This includes labels, file sequence, program size, etc.

   L. Final Assembly Date.

8. Prepare schedule card request and production release.

11.8. EXERCISE

Prepare a FAC schedule card indicating the need for 4ØØØ cells of IBANK,1ØØØØ cells of DBANK, two IIIA tapes, and no drum.

ANSWER

, FAC, DB/5, IB/2, MTØ/2: CT: DT: ET: FT: GT: PRNTT: PNCHT: READT , MDØ/Ø.

12. ASSEMBLY, CORRECTING AND DEBUGGING

12.1. ASSEMBLY CONTROL CARDS

The present system for assembly requires that certain cards be present at the beginning and the end of the FORTRAN statements. These cards are referred to as START, MAP, FOR, and STOP cards.

12.1.1. THE START CARD

The first card in a program element should be the START card of the following form:

Col. 1-7	Col. 13-18	Col. 31-32	Col. 73-78
/././.*	NAME	CF	REWINT

Note: Col. 73-78 may be blank if a number of program sections are to be stacked. However, the START card for the last section should contain "REWINT."

12.1.2. THE MAP CARD

The MAP card should follow the START card of the main program on an original assembly. Unless it is desired to change the program name, it is not required for reassemblies.

The form of the MAP card is:

@ MAP $name_1$, $name_2$.

where $name_1$ is the 1 to 6 character name of the ROC program produced by the FORELF run and should be the same as $name_2$ of the FOR Card.

$Name_2$ is the 6-character name of the main source deck and should be the same as $name_3$ of the FOR card.

The period following $name_2$ must be present.

"MAP" should start in column 3. The "@" should be in column 1.

12.1.3. THE FOR CARD

This card is required for all assemblies. It informs the system that this is a FORTRAN assembly. It follows the MAP card if present. Its form is:

@OPTIONS FOR $name_1$, $name_2$, $name_3$

where options are:

I print the source language

L produce complete listing with machine language

N suppress all except stated printing by the processor

P write the RB element onto PNCHT. This is a required option.

Z delete this element

Options may be coded as a string beginning at Column 2 (Example: @ILP FOR $name_1$, $name_2$, $name_3$).

Note: DO NOT FORGET TO CODE "FOR" AFTER THE OPTIONS.

$Name_1$ is the source program name (in the case of a reassembly, the same as the old name). It may be the same name as $name_2$.

$Name_2$ is the updated source name (the new name). This must be the same name as $name_1$ of the MAP card.

$Name_3$ is the name attached to the relocatable binary output of the compiler. This name must be the same as $name_2$ of the MAP card.

When external subroutines are included in an assembly, all names in the FOR card must be unique.

12.1.4. THE STOP CARD

This card must be placed following the last statement in the program source deck. It is comprised of six (6) right parentheses beginning in Column 1. It should be followed by three blank cards. Right parentheses may be more in number, but only 6 are required.

12.1.5. EXAMPLES OF CONTROL CARDS

12.1.5.1. Original assembly with internal and external subroutines as part of the same program.

```
/./.*    PROG    CF        REWINT
@ MAP PROG,PROX.
@ILP FOR PROG,PROG,PROX
     MAIN PROGRAM statements
     INTERNAL SUBROUTINE statements
     END

@ILP FOR SUB1,SUB1,SUBA
     EXTERNAL SUBROUTINE statements
     END
))))))
     3 blank cards
```

12.1.5. EXAMPLES OF CONTROL CARDS (Continued)

12.1.5.2. Reassembly with changes to main program and a listing of external subroutine.

```
/././.*    PROG      CF                REWINT
@ILP FOR PROG,PROG,PROX
-10,10
@ILP FOR SUB1,SUB1,SUBA
))))))
     3 blank cards
```

12.1.5.3. Original assembly to stack programs.

```
/././.*     FPCODE    CF
@ MAP FPCODE,FPCOD1.
@ILP FOR  FPCODE,FPCODE,FPCOD1
     program
))))))
3 blank cards
/././.*     LPCODE     CF                REWINT
@ MAP LPCODE,LPCOD1.
@ILP FOR LPCODE,LPCODE,LPCOD1
     program
))))))
     3 blank cards
```

12.1.5.4. Reassembly to change program name.

```
/././.*    NPROG          CF                REWINT
@ MAP NPROG,PRA.
@ILP FOR OPROG,NPROG,PRA
     any corrections
))))))
     3 blank cards
```

12.2. PROGRAM CORRECTIONS

Rather than resubmitting your entire program, with a few changes, every time a correction needs to be made to the latest coding, it is possible to alter only those lines which need to be changed.

Each line of the printed program has been assigned a number, starting with 1, which is to the immediate left of your coded statements and which is followed by a period (.). You must refer to this number to make program corrections.

All the corrections must be coded in numerical order.

There are two types of corrections which can be made: (1) lines may be deleted,and (2) lines may be added. To add a line, the number of the statement immediately preceding the area where the insert is desired is referenced. An entire card is used to reference this line number; a minus (-) is placed in column 1 of the card, and the line number immediately follows the minus. The lines to be added are placed on the subsequent punched cards. For example, if J = J + 1 is to be added after the tenth line of the original coding, the correction program would be as follows:

```
C          C              C              C
1         13             31             73
/././.*  PROG            CF             REWINT
@ILP FOR PROG, PROG, PROG2
-1Ø
     J = J + 1
))))))
```

_____+ 3 blank cards _____

If lines are to be deleted from the original program, the same general format is used; but after the minus the number of the first line to be deleted is mentioned, followed by a comma, followed by the number of the last line in that section to be deleted. For instance,

-1Ø, 2Ø

will delete all lines between and including line 1Ø and line 2Ø. Lines may replace these deleted ones by placing the new coding immediately after this deletion card as in the insertion method.

```
C          C              C              C
1         13             31             73
/././.*  GOOD1            CF             REWINT
@ILP FOR GOOD1, GOOD1, MASTER
-23
     INSERT CARD(S)
-41, 56
     INSERT CARD(S)
-60,62
-74
     INSERT CARD(S)
))))))
3 blank cards
```

12.2. PROGRAM CORRECTIONS (Continued)

The example will:

(1) Insert N number of cards following card number 23.

(2) Delete cards number 41 - 56 and insert N number of cards following card number 40.

(3) Delete cards number 60 - 62.

(4) Insert N number of cards following card number 74.

12.3 FORTRAN DEBUGGING

A number of areas must be assumed as prior knowledge prior to the discussion of debugging of any computer program. In FORTRAN for the Univac 1107, certain basic concepts (not generally developed as necessary to FORTRAN programers) must be developed prior to actual program debugging.

The Univac 1107 executes production under control of a master program - EXEC 1. This master program assigns each scheduled program to a specific area within the computer and designates which specific units of peripheral equipment will be used by the operating program for each run. The internal memory assigned by EXEC will normally be divided into two separate physical areas within the computer, designed as Instructions (IBANK) and Data (DBANK). The IBANK instructions are executed sequentially to effect the execution of the program, and the DBANK contains buffers, constants and similar information.

Three types of error situations are detected by the FORTRAN compiler and flagged on the FORTRAN compilation listing. These are: (a) Diagnostic messages, where it appears to the compiler that bad form or judgment may have been used, (b) Error messages, where it is obvious to the compiler that bad form has been used, and (c) Fatal Error messages, where the compiler is unable to interpret the source program.

Other types of errors, either clerical or logical, may be undetected by the FORTRAN compiler. These are the types of errors which are most difficult to locate and resolve. In some cases these errors are undetected until the programer reviews the result of several test runs. However, in many cases, they will cause the program to be terminated abnormally prior to the completion of his testing runs.

The EXEC program provides materials which can be helpful in locating this type of error. These are the program termination log -- prepared by EXEC when the program has reached termination, the program trouble dump, which accompanies the termination log if the termination was abnormal, and the message log (a listing of all typeins and typeouts for a program).

Abnormal terminations are divided into three classifications: Operator Termination, EXEC Termination, and Error Termination. The classification of errors will be noted on both the trouble dump and the termination log. Operator terminations may or may not be accompanied by a trouble dump. This class of termination is initiated by a console message from the operator in which the option "dump the program" or "do not dump the program" is included in the message. EXEC terminations are initiated when the EXEC program detects an illegal situation, instruction, or sequence.

12.3 FORTRAN DEBUGGING (Continued)

Error terminations are initiated by the program or its subroutines when an invalid situation is encountered and no recovery is possible or probable. In FORTRAN both EXEC and Error terminations provide a trouble dump.

The termination log provides certain amounts of control information. For debugging purposes the helpful entries will be: (a) Date and Wall Time, which show when your program was ended, (b) Facility Assignments, the areas of core and drum and the items of peripheral equipment which were initially assigned to your program. An additional time (PROCEED TIME) is indicated on the termination log. The difference between PROCEED TIME and Termination Log WALL TIME will indicate the time your program was running (Time Sharing) in the computer. The RUN TIME entry will indicate the time your program kept control. This entry will never be less than 00: 00: 01 if a PROCEED TIME is indicated. The termination log will indicate NORMAL, EXEC, ERROR, or OPER for the type of termination and the machine address where the last instruction was executed. If the termination is EXEC it will be followed by an additional error code indicating the type of error. Other types of terminations do not provide error codes for FORTRAN programs.

The program trouble dump is divided into three areas. The first is a visual octal image of the thin film control registers. The second and third are the octal contents of the IBANK instructions and the DBANK buffers, data and constants. These latter areas are designated as BANK1 and BANK2. Care should be used in transcribing the octal contents of these areas into Instructions. BCDXS3, Fieldata, Real or Integer values. The trouble dump illustrates eight memory locations per line, preceded by a six-position octal address for the first location on each line.

The instructions on your compilation listing are compressed, modified for relative loading, and appear as the locations in BANK1. To locate a comparable instruction:

If the initial address of BANK1	=	130000
Add the address on the compilation listing		000050
This is the memory location where the instruction is stored		130050

12.3. FORTRAN DEBUGGING (Continued)

The leftmost two octal digits at the trouble dump location and on the compilation listing should agree, as this is the operation code part of the instruction line. There probably are (or should be) many more locations shown on the trouble dump than are indicated on the compilation listing. These are the FORTRAN subroutines and/or FOLION locations.

The BANK2 values are assigned to consecutive locations as indicated by the DIMENSION statement, unless the programer elects the use of EQUIVALENCE or COMMON statements. The areas filled by DATA statements do not appear in sequential locations as referenced within the program, unless the DATA statements' reference areas were previously dimensioned. The labels of areas created by a DIMENSION or DATA statement are listed alphabetically on the compilation listing, under the heading "Storage Assignment for Variables." If a program contains only DIMENSION and DATA statements the location of these areas is found by adding in octal:

If the BANK2 starting address	=	060000
plus the Data Label address		002507
(From Storage Assignment for Variables)		
		062507

If a program uses the COMMON or EQUIVALENCE statement, the data areas mentioned in these statements are handled differently. During the ELF processing, the labels and size are listed independently on the ELF FACILITY SYNOPSIS, under the heading CORE. The locations are moved (in ELF listing order) to the initial positions in BANK2. All additional areas created by a DIMENSION or DATA Statement are displaced by the total number of core locations indicated on the ELF listing, excluding the entries IBANK, DBANK and $ERROR. If a program contains these statements, the location of DATA areas is found by adding in octal:

The BANK 2 starting address	060000
The number of locations indicated on the ELF listing	002400
The DATA LABEL (from Storage Assignment for VARIABLES)	001307
	063707

If a reference is made to LPM_n, then 156 octal locations must be added to the DBANK location. These locations occur at the beginning of the DBANK.

In the case of external subroutines, another factor, the relative address given in the ELF allocation, should be included in the computation to find the location desired.

12.3. FORTRAN DEBUGGING (Continued)

The 640-word area for IIIA buffer storage is divided as follows:

35	FORTRAN Control Words
1	LION Control Word
600	Data Words
4	LION Control Words
640	

The 155-word area for IIA buffer storage contains an initial 35 FORTRAN Control words and 120 Data words.

The 35 Control Words preceding the Buffer correspond to the SLEUTH programer's IICON for input files and IOCON for output files. Certain key words within this area may aid in debugging.

Word 1 - contains the file name in FIELDATA

Word 2 - left half contains the address of the file label/right half contains the number of label words

Word 8 - The EXEC status word. The left-most two octal digits contain the status of the read (or write) just completed. A code of ØØ is normal. A list of possible status codes is in the EXEC 1 Manual, UP-2577, Rev. 2, Appendix F.

Word 9 - The leftmost two octal digits indicate the function (READ, WRITE, REWIND, etc.) last performed or being processed. A list of these functions is in the EXEC I Manual, UP-2577, Rev. 2, pg. VI - 33 to 35.

The remainder of the word is a binary code referencing the channel and unit being used or the drum address referenced. If the remainder of this word is zero, no unit (or the incorrect unit) was scheduled for this file.

Word 28 - This word should contain the starting address of the current data record, if this is an input file. If no value is present (word contains zeros) then no data has been presented to the program. This is called the buffer position in the right half of the word.

Word 33 - Computed record count for input.
Cumulative record count (IIIA) for output.

12.3. FORTRAN DEBUGGING (Continued)

The thin film images can be helpful in tracing the execution of a program through its logical flow. Certain key registers can tell you where a program has been or what it has done and therefore give the programer a general idea of where to look on the compilation listing for clerical errors and inconsistencies.

$B0 - during an EXEC termination, the EXEC error code is placed in this register. (See 19.5 for Error Code). The left half of the register is used for the error; the right half for the address.

$B1 - in an ERROR termination, the contents of this register is one greater than the last instruction executed.

$B11 - an instruction address is placed in this register by each reference to I/O or CALL. The address in the register should be one (possibly more) greater than the last READ, WRITE, CALL, etc. executed.

$B12-$B15 - these registers contain the same values as $A0 - $A3

$A0 - in an ERROR termination, this register contains a negative value.

$R1 - this is a repeat counter for single instruction being repeated. The value in the register is the number of repetitions remaining. Use this in conjunction with $R3.

$R2 - a MASK register used to hold the extraction masks for formatting and similar operations.

$R3 - A temporary storage register, used to hold an instruction address during a repeated instruction. If ($R1) = 0, then ($R3) is one greater than the last repeated instruction. If ($R1)>0, then usually ($R3) is two greater than the last repeated instruction.

$R4 - $R15 -these registers are counters (indices) for repeated sequences - generally DO loops, where the loop generates more than one repeated instruction. The SLEUTH II coding on the right half of the compilation listing will indicate the following line near (preceding or following) the DO statement:

LR , XM R(n) , (i)

R(n) is the specific R register used

(i) is one less than the desired number of iterations

12.3. FORTRAN DEBUGGING (Continued)

Several unconnected DO's may use the same R register in different areas of the FORTRAN program. On the execution of a DO loop the number (i) is placed in R(n) and is decremented on each iteration to a final value of -1 (or (777777777776) octal). If the register contains zero, then no loop using that register was initiated or the loop was interrupted during the final iteration. A positive value less than (i) is the number of iterations remaining in the loop.

Notes in reference to memory dump addresses:

A memory dump of a program loaded in the upper core portion of the 1108--module 33 or higher--will reveal an octal difference of 200,000 between the stub addresses and addresses contained within a dumped word. This is also true of addresses contained within the registers. The stub address is an absolute address. Add an octal 200,000 to the location addresses contained within a dumped word or a register to obtain the absolute address referenced. The two exceptions to this rul are the absolute beginning addresses of the IBANK and DBANK which appear in the display of $ERROR+9 and $ERROR+10.

Under certain conditions, such as cross bank loading, the facility assignments in the Termination Log do not show the starting addresses of the IBANK and DBANK. To locate the starting addresses, use the following procedures:

1. Locate the EXEC assigned Facility line in the message log. An example of this line is:

```
CODE   *SORT  I14/03  D17/06  020270/1144  #
```

I14/03 indicates that the IBANK occupies three modules of core starting in module #14. D17/06 indicates that the DBANK occupies six modules of core starting in module #17.

2. To determine the starting address of either bank, convert the module number to its octal equivalent and multiply by octal 4000. For instance:

I14 is module 14. The octal equivalent is 16.

```
   16
x 4000
70000 octal
```

Last Reference to FOLION -

Last reference to a particular file may be found by examining word 31 of the file control table where $B11 is stored.

Last reference to FOLION made by any file may be found by adding octal 40 to the relative IBANK location of FOLION (see the ELF allocation) and adding the initial address of IBANK to find the instruction which stores $B11 in an address. Following the address where $B11 is stored is the octal constant 770077007700.

12.4. Composition of a FORTRAN IV Compilation Listing

The sample program which follows illustrates the "L" option listing generated by the FORTRAN IV compiler. If the "S" option had been used, no octal and mnemonic machine language code would appear. If the "N" option had been used, only the identification line and terminal messages would be printed.

The various elements listed are:

1. Time, date, machine, identification line
2. Start card line
3. "FOR" assembly line
4. Processor version line
5. Type of element (main program, subroutine, or function) and entry points line
6. Storage used section (block, name, length)
   - block - control counter for a section
   - name - *CODE for instruction area
     *DATA for non-COMMON data area
     *BLANK for blank-COMMON data area
     other names for labeled COMMON areas
   - length - (octal) number of storage location
7. External references (block, name)
   All external subprograms and FORTRAN library subroutines referenced are listed here with their block number and name.
8. Storage assignment MAP
   All names are sorted alphanumerically for easy reference.
   Information about each entry is in the following format:
   bbbT llllll NNNNNN.
   bbbb - is the number of the storage block containing the item
   T - is the type of item if it is not a label:
   - I - integer
   - R - real
   - D - double precision
   - C - complex
   - L - logical

   blank indicates an equivalenced or dimensioned variable that is not referenced in this element

   llllll - the relative octal location of the item in the main storage block.
   NNNNNN - The symbolic name of the item or label. If the item is a label, a letter (L, F, or G) follows the label number. L indicates a transfer label, F indicates a format label, and G indicates a transfer label generated by the compiler.

9. Source code listing
The major portion of the listing is comprised of the source program statements and the octal and mnemonic representations of the generated code. A line number (decimal), followed by a period, is assigned to each card image. These line numbers are referenced for updating a program. Another number (an internal sequence number in octal) is assigned to each group of information supplied to the compiler by the FORTRAN statement. More than one sequence number may be associated with one FORTRAN statement. The sequence numbers are printed down the left margin. Any diagnostic messages generated by the compiler are prefixed with the sequence number of the information in question. Diagnostics are interspersed among the FORTRAN statements.

The code generated to perform operations requested by a FORTRAN statement is generally listed following the statement. However, the optimization capabilities of the compiler may cause the code to be generated at a point other than that from which it was requested. The octal representation of the generated code is printed in UNIVAC 1107 instruction format. Each instruction is preceded by its relative octal location in the instruction area of the elements. Data values assigned at compilation time are printed in internal octal form. Each data item is preceded by its relative location in the block and followed by the block number. Both octal and mnemonic representations of the generated instructions are printed on the same line.

10. Terminal messages
Following the source listing, an end of compilation line is printed. On this line, the total number of diagnostic messages generated for the program unit is printed. If errors were detected in the generated code or if code generation was prohibited by a fatal error, a line indicating this is printed. If a fatal error occurs, the ELF processor (which generates relocatable code) cannot proceed and the assembly error terminates.

11. ELF processor messages
If a reassembly, the relocatable element (third name on FOR card) which was deleted and replaced by the new assembly is named. If any previously assembled relocatable elements is not reassembled; its name will appear and is still a part of the program. At the end of all elements the MAP statement is printed.

The ELF allocation map is printed with the names of all elements contained in a program and their respective IBANK and DBANK octal addresses. Following the ELF allocation map is the cross reference list of all entry points. Next is the list of I-O facilities for a FORTRAN assembly.

Then a summary of the total IBANK and DBANK requirements in core is expressed in decimal:

```
CORE    IBANK  ------ XXXXXX
        DBANK  ------ XXXXX
        $ERROR ------    97  [add to IBANK]
        $PARAM ------   110  [add to DBANK]
        N$BLNK   LN$BLN XXXXXX [blank COMMON]
           ?        ?   XXXXXX [named COMMON]
```

Drum facilities are named.

Finally, a facility statement is listed.
An example is:
** FAC,IB/4,DB/2,MDO/0,MAO/10,MTO/10,MCO/5.

This FAC statement is useful in preparing specification statements necessary to run under EXEC I. See 11.5. for instructions to prepare schedule cards.

Note. In the case of some errors, and always in the case of a fatal error, only the source program is listed.

```
/./.*     SAMPL          CF          #4100 CURTO          REWINT
BILD FOR SAMPL,SAMPL,SAMPX
COMPILATION BY CENSUS 1107 FORTRAN IV - SLEUTH II PROCESSOR.                    MODEL C5A


   MAIN PROGRAM          ENTRY POINT 000000

   STORAGE USED (BLOCK, NAME, LENGTH)

        0001    *CODE    000137
        0000    *DATA    000074
        0002    *BLANK   000024


   EXTERNAL REFERENCES (BLOCK, NAME)

        0003    TYPE
        0004    COMPUT
        0005    NRDU$
        0006    NIO1$
        0007    NIO2$
        0010    NWDU$
        0011    NSTOP$


   STORAGE ASSIGNMENT FOR VARIABLES (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000004 1L        0000   000006 10F       0001   000013 114G      0001   000032 125G      0001   000076 150G
    0001   000125 165G      0000   000014 20F       0000   000044 30F       0001   000060 35L       0001   000063 36L
    0001   000133 99L       0000 R 000001 A         0002 R 000010 B         0002 R 000022 C         0000 R 000002 D
    0000 R 000003 F         0002 R 000006 H         0002 I 000021 I         0000 I 000000 I%        0002 I 000013 J
    0000 R 000004 O         0002 R 000014 P         0000 R 000005 Q         0002 R 000015 R         0002 R 000016 S
    0002 R 000017 T         0002 R 000000 TAX       0002 R 000020 U         0002 R 000023 Z


00101     1.          COMMON TAX(6),H(2),B(4),J,P,R,S,T,U,I,C,Z
00103     2.          DATA TAX /.2,.2,.15,.1,.05,.0/

                      0002   000000   176631 463146
                      0002   000001   176631 463146
                      0002   000002   176463 146314
                      0002   000003   175631 463146
                      0002   000004   174631 463146
                      0002   000005   000000 000000
```

```
00106     4.      20 FORMAT(1H /,
00106     5.        A40H NUMBER       NAME          NO. HOURS  OV,
00106     6.        B40H.TIME GROSS PAY  OVER. PAY   TAXES   RET,
00106     7.        C40HIREMENT    INSUR.   NET PAY   PAY PERIOD//)
00107     8.      30 FORMAT (1X,4A6,4X,F4.1,6X,F4.1,4X,3(F5.2),6X,
00107     9.        A F5.2,7X,F4.2,6X,F5.2,5X,A6,A2/// )
00110    10.        CALL TYPE (14,14HSAMPLE PROGRAM)
00110                          000000   7413 13 00 0 000000   0003          LM.I      B11,TYPE
00110                          000001   0000 00 00 0 000064   0000          +         (14)
00110                          000002   0000 00 00 0 000060   0000          +         (H.A.)
00110                          000003   7406 00 00 0 000110                 NOP       A0,72
00111    11.         1 READ (5,10) A,B,C,D,I,F,J,H
00111                          000004   7413 13 00 0 000000   0005    1L    LM.I      B11,NRDU$
00111                          000005   0004 11 00 0 000065   0000          +         00111,(5)
00112                          000006   0000 00 00 0 000006   0000          +         10F
00112                          000007   1017 00 00 0 000001   0000          LA.XM     A0,A
00112                          000010   7201 02 00 0 000000   0006          SL.I      NIO1$
00112                          000011   2700 01 00 0 000066   0000          LX        B1,(1,1)
00112                          000012   2317 04 00 0 000002                 LR.XM     R4,2
00115                          000013   1017 00 01 2 000007   0002    1146  LA.XM     A0,B-1,*B1
00115                          000014   7201 02 00 0 000000   0006          SLJ       NIO1$
00116                          000015   7004 04 00 0 000013   0001          JGD       R4,114G
00116                          000016   1017 00 00 0 000022   0002          LA.XM     A0,C
00120                          000017   7201 02 00 0 000000   0006          SL.I      NIO1$
00120                          000020   1017 00 00 0 000002   0000          LA.XM     A0,D
00121                          000021   7201 02 00 0 000000   0006          SL.I      NIO1$
00121                          000022   1017 00 00 0 000021   0002          LA.XM     A0,I
00122                          000023   7201 01 00 0 000000   0006          SL.I      NIO1$
00122                          000024   1017 00 00 0 000003   0000          LA.XM     A0,F
00123                          000025   7201 02 00 0 000000   0006          SLJ       NIO1$
00123                          000026   1017 00 00 0 000013   0002          LA.XM     A0,J
00123                          000027   7201 01 00 0 000000   0006          SLJ       NIO1$
00123                          000030   2700 01 00 0 000066   0000          LX        B1,(1,1)
00123                          000031   2317 04 00 0 000001                 LR.XM     R4,1
00126                          000032   1017 00 01 2 000005   0002    125G  LA.XM     A0,H-1,*B1
00126                          000033   7201 02 00 0 000000   0006          SLJ       NIO1$
00126                          000034   7004 04 00 0 000032   0001          JGD       R4,125G
00131    12.           IF (I.EQ.99) GO TO 999
00131                          000035   7201 00 00 0 000000   0007          SL.I      NIO2$
00131                          000036   1000 00 00 0 000021   0002          LA        A0,I
00131                          000037   1517 00 00 0 000143                 ANA.XM    A0,99
00131                          000040   5100 00 00 0 000014                 TNZ       A0.
00133    13.           WRITE (6,20)
00133                          000041   7404 00 00 0 000133   0001          J         999L
00133                          000042   7413 13 00 0 000000   0010          LM.I      B11,NWDU$
00133                          000043   0005 13 00 0 000067   0000          +         00133,(6)
00133                          000044   0000 00 00 0 000014   0000          +         20F
00135    14.           0=F-40.0
00135                          000045   7201 00 00 0 000000   0007          SL.I      NIO2$
00135                          000046   1000 00 00 0 000070   0000          LA        A0,(4.000000+01)
00135                          000047   7601 00 00 0 000003   0000          FAN       A0,F
00136    15.           IF (0.LT.0.0) GO TO 35
00136                          000050   0200 00 00 0 000004   0000          SNA       A0,0
00136                          000051   1000 00 00 0 000004   0000          LA        A0,0
```

```
00136                                000052  7402 00 00 0 000055  0001                JP      A0,$+3
00136                                000053  4400 00 00 0 000071  0000                TEP     A0,(040040000000)
00140    16.          Z=40.0
00140                                000054  7404 00 00 0 000060  0001                J       35L
00140                                000055  1000 00 00 0 000070  0000                LA      A0,(4.000000+01)
00141    17.          GO TO 36
00141                                000056  0100 00 00 0 000023  0002                SA      A0,Z
00141                                000057  7404 00 00 0 000063  0001                J       36L
00142    18.       35 O=0.0
00143    19.          7=F
00143                                000060  0500 00 00 0 000004  0000       35L      SZ      0
00143                                000061  1000 00 00 0 000003  0000                LA      A0,F
00143                                000062  0100 00 00 0 000023  0002                SA      A0,Z
00144    20.       36 CALL COMPUT (Q,D)
00144                                000063  7413 13 00 0 000000  0004       36L      LMJ     B11,COMPUT
00144                                000064  0000 00 00 0 000005  0000                +       Q
00144                                000065  0000 00 00 0 000002  0000                +       D
00145    21.          WRITE (6,30) A,B,F,D,P,Q,R,S,T,U,H
00145                                000066  7406 00 00 0 000144                      NOP     A0,100
00145                                000067  7413 13 00 0 000000  0010                LMJ     B11,NWOU$
00145                                000070  0006 05 00 0 000067  0000                +       00145,(6)
00146                                000071  0000 00 00 0 000044  0000                +       30F
00146                                000072  1017 00 00 0 000001  0000                LA,XM   A0,A
00146                                000073  7201 02 00 0 000000  0006                SLJ     NIO1$
00146                                000074  2700 01 00 0 000066  0000                LX      B1,(1,1)
00146                                000075  2317 04 00 0 000002                      LR,XM   R4,2
00151                                000076  1017 00 01 2 000007  0002       150G     LA,XM   A0,B-1,*B1
00151                                000077  7201 02 00 0 000000  0006                SLJ     NIO1$
00152                                000100  7004 04 00 0 000076  0001                JGD     R4,150G
00152                                000101  1017 00 00 0 000003  0000                LA,XM   A0,F
00154                                000102  7201 02 00 0 000000  0006                SLJ     NIO1$
00154                                000103  1017 00 00 0 000002  0000                LA,XM   A0,D
00155                                000104  7201 02 00 0 000000  0006                SLJ     NIO1$
00155                                000105  1017 00 00 0 000014  0002                LA,XM   A0,P
00156                                000106  7201 02 00 0 000000  0006                SLJ     NIO1$
00156                                000107  1017 00 00 0 000004  0000                LA,XM   A0,Q
00157                                000110  7201 02 00 0 000000  0006                SLJ     NIO1$
00157                                000111  1017 00 00 0 000005  0000                LA,XM   A0,R
00160                                000112  7201 02 00 0 000000  0006                SLJ     NIO1$
00160                                000113  1017 00 00 0 000015  0002                LA,XM   A0,S
00161                                000114  7201 02 00 0 000000  0006                SLJ     NIO1$
00161                                000115  1017 00 00 0 000016  0002                LA,XM   A0,T
00162                                000116  7201 02 00 0 000000  0006                SLJ     NIO1$
00162                                000117  1017 00 00 0 000017  0002                LA,XM   A0,U
00163                                000120  7201 02 00 0 000000  0006                SLJ     NIO1$
00163                                000121  1017 00 00 0 000020  0002                LA,XM   A0,U
00163                                000122  7201 02 00 0 000000  0006                SLJ     NIO1$
00163                                000123  2700 01 00 0 000066  0000                LX      B1,(1,1)
00163                                000124  2317 04 00 0 000001                      LR,XM   R4,1
00166                                000125  1017 00 01 2 000005  0002       165G     LA,XM   A0,H-1,*B1
00166                                000126  7201 02 00 0 000000  0006                SLJ     NIO1$
00166                                000127  7004 04 00 0 000125  0001                JGD     R4,165G
00171    22.          GO TO 1
00171                                000130  7201 00 00 0 000000  0007                SLJ     NIO2$
```

```
00172  *DIAGNOSTIC*  CONTROL WILL NEVER REACH FOLLOWING STATEMENT.
00172    23.         GO TO 1
00172                              000131   7404 00 00 0 000004   0001              J       1,
00172                              000132   7404 00 00 0 000004   0001              J       1L
00173    24.     999 STOP
00173                              000133   7413 13 00 0 000000   0011        999L  LM,I    B11,NSTOP$
00173                              000134   0000 00 00 0 000072   0000              +       (005050505050S)
00173                              000135   7413 13 00 0 000000   0011              LM,I    B11,NSTOP$
00174    25.         END
00176                              000136   0000 00 00 0 000073   0000              +       (01223112527l4)

      END OF LISTING.      1 *DIAGNOSTIC* MESSAGE(S).
```

```
@ ELT SAMPX,3
 DELETED

@ILP FOR COMPUT,COMPUT,SUB1

COMPILATION BY CENSUS 1107 FORTRAN IV - SLEUTH II PROCESSOR.                    MODEL C5A


   SUBROUTINE COMPUT    ENTRY POINT 000052

  STORAGE USED (BLOCK, NAME, LENGTH)

      0001   *CODE   000062
      0000   *DATA   000007
      0002   *BLANK  000024


 STORAGE ASSIGNMENT FOR VARIABLES (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000036 SOL        0001   000041 60L        0001   000043 70L        0002   000010 B          0002 R 000022 C
    0002   000006 H          0002 I 000021 I          0002 I 000013 J          0002 R 000014 P          0002 R 000015 R
    0002 R 000016 S          0002 R 000017 T          0002 R 000000 TAX        0002 R 000020 U          0002 R 000023 Z


00101     1.          SUBROUTINE COMPUT (Q,P)
00103     2.          COMMON TAX(6),H(2),B(3),J,P,R,S,T,U,I,C,Z
00104     3.          Q=P*Q
00104                           000000   1000 00 00 0 000004   0000    LA      A0,NTEMP$+1
00104                           000001   7602 00 00 0 000001            FM      A0,Q
00105     4.          P=C*Z+Q
00105                           000002   0100 00 00 0 000001            SA      A0,Q
00105                           000003   1000 02 00 0 000023   0002     LA      A2,Z
00105                           000004   7602 02 00 0 000022   0002     FM      A2,C
00105                           000005   7600 02 00 0 000001            FA      A2,Q
00106     5.          I=I+1
00106                           000006   0100 02 00 0 000014   0002     SA      A2,P
00106                           000007   1000 04 00 0 000021   0002     LA      A4,I
00106                           000010   1417 04 00 0 000001            AA,XM   A4,1
00107     6.          IF (I.GT.5) I=6
00107                           000011   0100 04 00 0 000021   0002     SA      A4,I
00107                           000012   1000 00 00 0 000021   0002     LA      A0,I
00107                           000013   1517 00 00 0 000005            ANA,XM  A0,5
00107                           000014   5000 00 00 0 000014            TZ      A0.
00107                           000015   6000 00 00 0 000014            TP      A0.
00110                           000016   7404 00 00 0 000021   0001     J       NBPRO$+17
00110                           000017   1017 00 00 0 000006            LA,XM   A0,6
00111     7.          R=TAX(I)*P
00111                           000020   0100 00 00 0 000021   0002     SA      A0,I
00111                           000021   2700 01 00 0 000021   0002     LX      B1,I
00111                           000022   1000 00 00 0 000014   0002     LA      A0,P
```

```
00111                                  000023  7602 00 01 0 177776  0002                FM      A0,TAX-1,B1
00112     8.        S=P*.03
00112                                  000024  0100 00 00 0 000015  0002                SA      A0,R
00112                                  000025  1000 02 00 0 000000  0000                LA      A2,(3.000000-n2)
00112                                  000026  7602 02 00 0 000014  0002                FM      A2,P
00113     9.        IF (J-1) ,50,60
00113                                  000027  0100 02 00 0 000016  0002                SA      A2,S
00113                                  000030  1000 04 00 0 000013  0002                LA      A4,J
00115                                  000031  1417 04 00 3 177776                      AA,XM   A4,-1
00115                                  000032  7400 04 00 0 000036  0001                JZ      A4,50L
00116     10.       T=0.0
00116                                  000033  7402 04 00 0 000041  0001                JP      A4,60L
00117     11.       GO TO 70
00117                                  000034  0500 00 00 0 000017  0002                SZ      T
00117                                  000035  7404 00 00 0 000043  0001                J       70L
00120     12.   50  T=1.75
00120                                  000036  1000 00 00 0 000001  0000      50L       LA      A0,(1.750000+n0)
00121     13.       GO TO 70
00121                                  000037  0100 00 00 0 000017  0002                SA      A0,T
00121                                  000040  7404 00 00 0 000043  0001                J       70L
00122     14.   60  T=2.85
00122                                  000041  1000 00 00 0 000002  0000      60L       LA      A0,(2.850000+n0)
00122                                  000042  0100 00 00 0 000017  0002                SA      A0,T
00123     15.   70  U=P-R-S-T
00123                                  000043  1000 00 00 0 000015  0002      70L       LA      A0,R
00123                                  000044  7601 00 00 0 000014  0002                FAN     A0,P
00123                                  000045  7600 00 00 0 000016  0002                FA      A0,S
00123                                  000046  7600 00 00 0 000017  0002                FA      A0,T
00123                                  000047  0200 00 00 0 000020  0002                SNA     A0,U
00123                                  000050  2700 01 00 0 000005  0000      EXIT      LX      B1,NTEMP$+2
00123                                  000051  7404 00 13 0 000003                      J       3,B11
00123                                  000052  0600 01 00 0 000005  0000      ENTRY     SX      B1,NTEMP$+2
00123                                  000053  1000 00 13 0 000000                      LA      A0,0,B11
00123                                  000054  0101 00 00 0 000005  0001                SA,H2   A0,NBPRO$+5
00123                                  000055  0101 00 00 0 000001  0001                SA,H2   A0,NBPRO$+1
00123                                  000056  0101 00 00 0 000002  0001                SA,H2   A0,NBPRO$+2
00123                                  000057  2300 01 13 1 000001                      IR      R1,*1,B11
00123                                  000060  0400 01 00 0 000004  0000                SR      R1,NTEMP$+1
00124     16.       RETURN
00125     17.       END
00127                                  000061  7404 00 00 0 000000  0001                J       NBPRO$

END OF LISTING.     0 *DIAGNOSTIC* MESSAGE(S).
```

```
@ ELT SUB1,3
  DELETED
@ MAP SAMPL,SAMPX.
```

FLF	ALLOCATION SAMPL /FB 400	* RELATIVE ADDRESS IBANK	DBANK
	SAMPX /FB 400	000000	000000
	NPAUS$/MR1511	000137	000074
	NOUT$ /MR1511	000214	000076
	NTER$ /DC1011	001526	000234
	NINPT$/DC1611	001625	000244
	SUB1 /FB 400	002675	000351
	TYPE	002757	
	MEXIT$	003030	
	ERR	003112	
	NETV$ /MR0321	003170	
	NEMT$ /MR0416	003214	000360
	CONCH$/MR0321	004213	
	PRINT$/MR0321	005670	
	NTRL$	010010	
	CREAD$/MR0321	010115	
	******	011573	000617

ELF FACILITY SYNOPSIS * SAMPL *

```
S.LIST    SAMPX

EX.REF    SAMPX
          TYPE
          COMPUT
          NRDU$
          NIO1$
          NIO2$
          NWDU$
          NSTOP$

IO-FAC    PNCHA    MA    0      OP
          PRNTA    MA    0      OP
          GA       MA    0      OP
          FA       MA    0      OP
          EA       MA    0      OP
          DA       MA    0      OP
          CA       MA    0      OP
          BA       MA    0      OP
          AA       MA    0      OP
          READA    MA    0      OP
          PNCHT    MT    0      OP
          PRNTT    MT    0      OP
          GT       MT    0      OP
          FT       MT    0      OP
          ET       MT    0      OP
          DT       MT    0      OP
          CT       MT    0      OP
          BT       MT    0      OP
          AT       MT    0      OP
          READT    MT    0      OP
          PNCHC    MC    0      OP
          PRNTC    MC    0      OP
          BC       MC    0      OP
          AC       MC    0      OP
          READC    MC    0      OP

CORE      IBANK    -----   "087
          DBANK    -----   399
          $ERROR   -----    97
          $PARAM   -----     0
          N$BLNK   LN$BLN   20

DRUM      D1       LD1       0
          D2       LD2       0
          D3       LD3       0
          D4       LD4       0
          D5       LD5       0
```

** FAC,TB/3,DB/1,MD0/0,MA0/10,MT0/10,MC0/5.

13. THE TAPE SORT

13.1. USING FORTRAN GENERATED SUBPROGRAMS AS OWN CODING TO THE 1107 SORT MERGE

13.1.1. CONCEPT OF "OWN CODING"

The 1107 Sort/Merge is an independent program which, under certain circumstances, may be instructed to "call upon" subprograms to manipulate the data as it passes through the S/M process. These subprograms (if present) are called from the S/M at two points:

(1) OCFP - After an item is read from the input source by the S/M, but before it is presented to the sorting process, the "Own Coding First Pass" (if present) is given control. This subprogram may then edit or rearrange each item, change its length, reject it, add items from another source, prepare tables, etc.

(2) OCLP - After the input and sorting process is complete and while the sequenced items are being written onto the final output, the "Own Coding Last Pass" (if present) is given control before each item is written. This subprogram may also edit or rearrange each item, change its length, reject it, add items from another source, prepare tables, etc.

Although the OCFP and OCLP subprograms both become part of a single S/M program when they are loaded into the machine, they should be thought of as two completely separate programs. They are executed at two separate points in time and they may not communicate with each other except as explained later. However, one tape is used for both assemblies.

An initial typeout indicating the FORTRAN own coding name (the main program name is always SORT) and version number is recommended since control is more difficult when processing a program of this type.

"Own coding" may also be used with a special version of SORT called SORT4C. This version handles IBM Compatible Tapes in addition to all the units available in SORT. See Part 14-B-1 SORT4C of the Census Computer Systems Manual for parameter details and storage requirements.

13.1.2. CODING THE OCFP AND OCLP SUBPROGRAMS

```
/./.*     FPCODE
@ MAP FPCODE,FPCOD1.
@LIP FOR FPCODE,FPCODE,FPCOD1
        EXTERNAL FPSUB
1       CALL OCSUB ($1,$1,FPSUB)

        SUBROUTINE FPSUB ($,$,$)
        COMMON NENTRY,ITMSIZ,ITEM(20)
        IF (NENTRY) 100,200,300
           *
           *
        (YOUR OCFP FORTRAN STATEMENTS ARE CODED HERE)
           *
           *
700     RETURN 1
800     RETURN 2
900     RETURN 3
        END
))))))
3 blank cards

/./.*     LPCODE                              REWINT
@ MAP LPCODE,LPCOD1.
@LIP FOR LPCODE,LPCODE,LPCOD1
        EXTERNAL LPSUB
1       CALL OCSUB ($1,$1,LPSUB)

        SUBROUTINE LPSUB ($,$,$)
        COMMON NENTRY,ITMSIZ,ITEM(20)
        IF (NENTRY) 100,200,300
           *
           *
        (YOUR OCLP FORTRAN STATEMENTS ARE CODED HERE)
           *
           *
700     RETURN 1
800     RETURN 2
900     RETURN 3
        END
))))))
3 blank cards
```

Although the names in your actual coding need not be the same as those in this example, they must maintain the same relationship and consistency as shown in this example. Note that OCSUB is a constant name. It is the name of a special subroutine. Also note that NENTRY,ITMSIZ, and ITEM should occupy the first positions in the blank COMMON block.

13.1.3. ENTRANCES TO YOUR OWN CODING FROM THE S/M.

Statement No.
Shown in the
Examples

EXPLANATION

1ØØ This is the "write only return" from the S/M. This means that you previously took the RETURN 2 entrance to the S/M. The S/M is not presenting an item to the Own Coding in this case.

2ØØ This is the "end-of-file return" from the S/M. The S/M has no more items to present to you. Execution of the RETURN 1 statement will cause control to revert to the S/M, not to reenter this particular Own Coding subprogram again. Items may be added at this point by use of the RETURN 2 statement; a limited number only.

3ØØ This is the "read return" from the S/M. The S/M is presenting the Own Coding with an item which is contained in ITEM (1) to ITEM (n) while the length of that item is contained in ITMSIZ. Note that, in the coded example, ITEM was dimensioned with 20 locations. This will, of course, vary according to the size of the item expected. The "read return" from the S/M is the first entrance used to enter your Own Coding subprogram.

13.1.4. RETURNS FROM YOUR OWN CODING TO THE S/M.

RETURN 1 Your Own Coding elects not to present an item to the S/M but requests that the S/M present the Own Coding with another item.

RETURN 2 Your Own Coding has an item to present to the S/M but requests that the S/M not present another item to the Own Coding. Next entrance to your Own Coding will be through the "write only return" (statement number 100 in this example). The item that your Own Coding is presenting to the S/M is contained in Item (1) to ITEM (n), while the item length (n) is contained in ITMSIZ.

RETURN 3 Your Own Coding has an item to present to the S/M and requests that the S/M present another item to your Own Coding. Next entrance to your Own Coding will be through the "read return" (statement number 300 in this example). The item that your Own Coding is presenting to the S/M is contained in ITEM (1) to ITEM (n), while the item length (n) is contained in ITMSIZ.

13.1.5. I/O STATEMENT WITHIN THE OWN CODING SUBPROGRAMS

FORTRAN read/write statements may not be used within the Own Coding subprograms. At the present time, only FOLION and EX I/O Statements are permissible.

13.1.6. COMMUNICATION BETWEEN OCFP AND OCLP SUBPROGRAMS

Although the OCFP and OCLP subprograms should be thought of as two separate programs, all I/O devices are shared by them. Data written on drum or tape by OCFP is available to the OCLP subprogram. However, the FOLION package, if used, does not maintain definitions of opened files in both subprograms. Therefore, all files must be closed at the end of OCFP and then may be reopened at the beginning of the OCLP subprogram. For example, OCFP could write with FOLION on unit 10, and then close file with no rewind. OCLP could then reopen the file on unit 10 and continue writing on the same tape. Or, as an additional example, OCFP could write on unit 10, close with rewind, and then OCLP could reopen and read the file on unit 10. The same comments are applicable to drum.

In addition, a single named COMMON area is available to both OCFP and OCLP subprograms. Other than the blank COMMON statement shown in the coded example, the following is the only COMMON area which may be defined.

```
COMMON /OCTAB/ .....
```

Data placed in the OCTAB COMMON area by the OCFP subprogram will be available to the OCLP subprogram.

13.2. SCHEDULING

13.2.1. PTY CARD

The program name is SORT.
When there is no option coding, the user may elect to use SORTN.

13.2.2. FAC CARD

IBANK: Let x = the larger result of these two computations { OCFP IBANK - 7ØØ ; OCLP IBANK - 11ØØ }

then, the total IBANK requirement is: x + 6144 cells

DBANK: Let x = the larger DBANK requirement of either the OCFP or OCLP subprograms.

Then, the total DBANK requirement is:
X + COREWK + LOCTAB + LN$BLN + 4096

where COREWK, LOCTAB, and LN$BLN represent TAL card increments explained later.

MDØ/1ØØØØ Additional drum must be assigned to cover the sum of LGTHWK, the length of OCFP (both banks), the length of OCLP (both banks) and any drum used by the Own Coding subprograms (represented by the sum of the TAL card entries LD1 through LD5). (LGTHWK represents a TAL card increment explained later.)

MA1 This designates a channel which contains three 2A tape units. The unit tags are SM1 and SM2 with SM1A for the alternate I/O unit. (If the channel is not used, it should not be mentioned on the FAC card. Units contained on a channel which has been mentioned must be deleted if they are not to be used). For example, if we wished to schedule the 2A unit SM1, the FAC card entry would appear as follows:

```
,MA1/1: SM2:SM1A,
```

the above comments apply to all tape channels.

MA2 This designates a channel which contains three 2A tape units. The unit tags are SMA1 and SMA2 with SMA1A for the alternate I/O unit.

MA8 This designates a channel which contains the ten standard FORTRAN 2A tape units. The unit tags are AA through GA, READA, PRNTA, and PNCHA. These are the units which are scheduled if the Own Coding subprograms use 2A tapes directly.

MC9 This channel is used for 4C tapes AC, BC, PRNTC, PNCHC, and READC. These units are scheduled when FORTRAN Own Coding subprograms use 4C tapes directly as in the EX subroutine. When the sort's input or output is a 4C tape, then SORT4C should be used.

13.2.2. FAC CARD (Continued)

MT3 This designates a channel which contains eight 3A tape units. The unit tags are SMT1 through SMT7 with SMT1A for the alternate I/O unit.

MT6 This designates a channel which contains seven 3A tape units. The unit tags are SMTA1 through SMTA6 with SMTA1A for the alternate I/O unit.

MT7 This designates a channel which contains the ten standard FORTRAN 3A tape units. The unit tags are AT through GT, READT, PRNTT, and PNCHT. These are the units which are scheduled if the Own Coding subprograms use 3A tapes directly.

13.2.3. TAL CARD

COREWK This represents the number of data bank core locations to be used for working storage by the sort.

LGTHWK This represents the number of drum locations to be used for string storage by the sort.

LN$BLN This entry is equal to the value which appears on the ELF facility synopsis opposite "LN$BLN." If both OCFP and OCLP subprograms are present, the TAL entry should be the larger of the values appearing on the two ELF listings.

LOCTAB This represents the length of the OCTAB COMMON area, if used. This entry is equal to the value which appears on the ELF facility synopsis opposite "LOCTAB." If both OCFP and OCLP subprograms are present, the TAL entry should be the larger of the values appearing on the two ELF listings.

LD1 through LD5 These represent the Own Coding drum table lengths, if used. They correspond to FORTRAN internal unit numbers 25 through 29.

13.2.4. SUB CARD

For OCFP only, the sub card should be:

JRID, SUB, OCFP/FPCOD1, OCLP.

where FPCOD1 is the same as the second name on the MAP card of the OCFP subprogram.

For OCLP only, the sub card should be:

JRID, SUB, OCFP, OCLP/LPCOD1.

where LPCOD1 is the same as the second name on the MAP card of the OCLP subprogram.

For OCFP and OCLP, the sub card should be:

JRID, SUB, OCFP/FPCOD1, OCLP/LPCOD1.

where FPCOD1 and LPCOD1 are as described above.

It is possible to use SORT with no option coding. In this case, the SUB card should be:

JRID, SUB, OCFP, OCLP.

When SORTN is used, no SUB card is required.

13.2.5. PMO CARD

Column	No. of Characters	Parameter	Coding
12	1	Sort or Merge	Blank = Sort 1 = Merge 2 = Merge/message (13.4)
13	1	Ascending or Descending Sequence	Blank = Ascending 1 = Descending
14	1	Whether or not to use machine sequence ($ØØ$-77_8)	Blank = Machine Sequence 1 = Different Sequence
15	1	Fixed or variable length items	Blank = Fixed 1 = Variable
16	1	Rearrangement of keys	Blank = Rearrangement 1 = No Rearrangement
17	1	Type of key Note: When column 13 indicates descending sequence, use type of keys under ascending to produce descending sequence and types 5-9 to produce ascending sequence.	Ascending: Blank = Unsigned binary 1 = Signed decimal 2 = Unsigned binary 3 = Signed binary (11Ø7) 4 = Signed binary (IBM) Descending: 5 = Unsigned binary 7 = Signed decimal 8 = Signed binary (11Ø7) 9 = Signed binary (IBM) Ascending/Descending: 6 = Combination of key types.
18	1	Must be blank	Blank
19	1	Location of input	1 = Uniservo IIA (Census Compatible) 3 = Uniservo IIIA 6 = Drum
2Ø	1	Input file label	Blank = Check input tape label against the label specified on the PM3, 4, and 5 cards. 1 = Do not check labels

13.2.5. PMO CARD (Continued)

Column	No. of Characters	Parameter	Coding
21	1	2/Output file label	Blank = Use contents of PM6, 7, and 8 cards as output label. 1 = Use input tape label as output tape label.
22	1	Type of intermediate storage	3 = Uniservo IIIA 6 = Drum 9 = Drum backed by Uniservo IIIA
23	1	Location of output	1 = Uniservo IIA (Census Compatible) 3 = Uniservo IIIA 6 = Drum
24-29	6	Maximum number of words in item	Decimal value
30-35	6	Number of bits in total key	Decimal value
36-41	6	Maximum number of words per input block	Decimal value *
42-47	6	Maximum number of words per output block	Decimal value *
48-49	2	Number of ways for merge (for straight merge only)	Decimal value
50-51	2	Total number of I/O tape units available excluding alternate - at least two	Decimal value

2/ When both input and output are in LION INTERNAL FORMAT free words (a maximum of 32), if present in the input tape label, will be transferred to the output tape label.

* This decimal value is the block size written on tape. It does not include the 35 words of file control information which FORTRAN programers need when using FOLION.

13.2.5. PMO CARD (Continued)

Column	No. of Characters	Parameter	Coding
52-53	2	Number of input tapes	Decimal value
54	1	Action required for unreadable blocks	Blank = Type block number and terminate 1 = Skip unreadable blocks 2 = Skip unreadable blocks but type block numbers
55	1	Own Coding First Pass	Blank = No 1 = Yes
56	1	Own Coding Last Pass	Blank = No 1 = Yes
57	1	Rerun	Blank = No 1 = Yes
58	1	Input Tape Format	Blank = Internal 1 = External
59	1	Output Tape Format	Blank = Internal 1 = External
60	1	PM3-PM5 convert FD to XS3	1 = convert, Blank = do not
61-64	4	Sort file fixed length item size	Decimal value 3/
65-68	4	Final output fixed length item size	Decimal value 3/
71	1	Lang PM6/ 0=FD;1=Convt;2=XS3 equiv.	Blank = Fieldata 1 = Fieldata to XS3
72	1	Language of input label	Blank = XS3 1 = Fieldata
73	1	Total number of IIA (Census Compatible) input tapes to Sort on a one for one basis	Decimal value
75	1	Whether or not to sort every file of a multi-file input	Blank = No 1 = Yes
76-77	2	File to begin sorting if a multi-file input	Decimal value

3/ Parameter need not be entered if the item size is the same as specified for the input file in columns 24-29.

13.2.5. PMO CARD (Continued)

RELATIONSHIP BETWEEN CARD COLUMNS 24-29, 61-64, AND 65-68 OF THE PMØ CARD

A = Columns 24-29 Input Item Size

B = Columns 61-64 Sort File Item Size

C = Columns 65-68 Final Output Item Size

13.2.6. PM1 CARD

This card is to be used to arrange fields upon which the items are to be sorted or merged. List major fields first, etc., down to the minor field.

Column	No. of Characters	Parameter	Coding
12-14	3	Word in item where first bit of field will be found*	Decimal value
15	1	Type of Key, if PMØ card specified a combination of Key types	See column 17 in PMØ format
16-17	2	Bit position in word of first bit in field	Decimal value
18-20	3	Number bits in the field	Decimal value

Repeat preceding four entries six more times if necessary.

PM2 CARD

PM2 is used to establish the sequence if the machine sequence is not desired. (Col. 14 of the PMØ card)

Beginning with column 12, 64 characters in the desired sequence are punched.

* The first word in an item is word 1.

13.2.7. PM3,PM4,PM5 CARDS

These cards are used to check the Input File label if this option has been selected in the PMØ CARD.

Column	No. of Characters	Parameter	Coding
12-17	6	Number of words in the File Label, including the date and reel number (maximum value is 32)	Decimal value, with either leading zeros or blanks
18-23	6	Date (Month, Day, Year)*	Numeric, e.g., 052064, if blank, date is not checked.
24-29	6	Reel Number*	Decimal value. If blank, the reel number is not checked.
30-77 12-77 (PM4 or PM5 Card)	6-180	File Label. It can be continued on PM4 and PM5, beginning with column 12	Alphanumeric

*If a IIA tape is read, there will be no date and reel number to be checked, so columns 18-29 should be blank.

For External Format, the PM3 card should contain:

Columns		
12-17	Word Count	Always 3 in column 17
18-23	Reel Number	Decimal value
24-35	File Name	Alphanumeric, space filled

13.2.8. PM6,PM7,PM8 CARDS

These cards are used to supply the Output File Label, if this option has been selected in PMØ.

Column	No. of Characters	Parameter	Coding
12-17	6	Number of words in the File Label, including the Date and Reel Number (maximum value is 32)	Decimal value with either leading zeros or blanks
18-23	6	Date (Month,Day,Year)*	Number, e.g., 052064 If blank, the run date, provided by EXEC 1 will be written.
24-29	6	Reel Number*	Decimal value. If blank, the first reel of the output will be set to 1.
30-77	6-180	File Label. It can be continued on PM7 and PM8, beginning with column 12.	Alphanumeric.

*If a IIA tape is written, no date and reel number will be written, so columns 18-29 should contain the first two words of the file label.

For External Format, the PM3 card should contain:

Columns		
12-17	Word Count	Always 3 in column 17
18-23	Reel Number	Decimal value
24-35	File Name	Alphanumeric, space filled

13.2.9. EXAMPLE OF SORT SCHEDULE CARDS

Description of problem:

25,000 120 word items.

Input is 3 IIIA's with 605 word blocks.

Output is to be on IIIA with 605 word blocks.

Sort keys are word 120, bits 5-0; word 97, bits 29-0; word 3, bits 35-0; word 4, bits 35-0; word 5, bits 35-0.

No input label checking.

Output label is supplied in XS3.

User's own coding for both first and last pass.

One IIIA tape is used in option coding.

The own coding first pass uses:

IBANK - 300

DBANK - 100

The own coding last pass uses:

IBANK - 6000

DBANK - 2000

FORM BC-1522e (4-6-67)

TEST
1107 EXECUTIVE SCHEDULE CARDS
FIXED PARAMETERS ONLY

U. S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

NOTE: Do not use this form for programs requiring more than one set of parameter cards. Use forms BC-1522b and BC-1522c

PROCESSING DIVISION USE ONLY

A. Division Code	B. Project Number	C. Project name	D. Programmer	E. Office telephone ext.
53	00833000000			

F. Name of program (If general purpose service routine such as sort, enter suggested JRID.)	G. Job identification (work unit, group, State, period, etc.)	H. Request status	I. Date of request	J. Time of request
JOBNAM		☐ Original ☐ Replacement		

Do not make entries in shaded columns. For all other columns, indicate the exact characters to be punched. Contents must be in accordance with EXECutive specifications.

JRID	TYPE	CC	PROGRAM NAME	INT	DMP	MEDIUM NAME	PRT	PTY	MXW	TIME	LIB. NAME
	PTY		SORT		T	*MASTER	P		P	15	USER

JRID	TYPE	FACILITY DESCRIPTIONS
	FAC	DB/10,IB/6,MT3/6:SMT1A:SMT7,MDO/7781500
	FAC	MT7/1:BT:CT:DT:ET:FT:GT:READT:PRNTT:PNCHT

JRID	TYPE	TABLE LENGTH INCREMENTS
	TAL	COREWK/15115,LGTHWK/759750

JRID	TYPE	RC	TRANSFERS
	TRN		
	TRN		

JRID	TYPE	OCFP & OCLP NAME
	SUB	OCFP/CP1,OCLP/CP2

JRID	TYPE	PARAMETERS
	PM0	31 93 120 144 605 605 6 3 11 1
	PM1	120 5 6 97 29 30 3 35108
	PM2	
	PM3	
	PM4	
	PM5	
	PM6	6 THIS IS AN OUTPUT TAPE
	PM7	
	PM8	
	PM9	
	PM0	53008330000000

CENFOR 13-16 1970

March 1970

USCOMM-DC

13.3. DETERMINATION OF CORE, DRUM, AND TAPE REQUIREMENTS FOR A SORT

Core working storage (COREWK), drum tables for storage of sorted strings (LGTHWK) and intermediate tapes must be assigned in the proper relationship to complete a sort operation in one merge pass. Excessive running time and a high incidence of error terminations will result if this is not done.

A simple formula will produce the most strategic assignment for LGTHWK and intermediate tapes. Then, COREWK may be determined by dividing LGTHWK by 50.

The formula for LGTHWK is:

$$LGTHWK = \frac{1}{T}\left[D \left(\frac{I+1}{I}\right) + 15{,}000 \right]$$ where

D = total number of data words

I = number of words per item

T = number of intermediate tapes, initially assigned as 1.

$$COREWK = \frac{LGTHWK}{50}$$ (must be at least 2048 words)

If the total number of data words exceeds 4,000,000 the sort should be done in parts, followed by a merge. A maximum of 4,500,000 data words can be sorted in one operation within the standards for facility assignment providing the record size is not less than 15 words and there is no option coding. The minimum configuration of intermediate tapes, if any are used, is 3. If the output tape is a IIIA tape, then the output tape can be considered one of the intermediate tapes.

13.3. DETERMINATION OF CORE, DRUM, AND TAPE REQUIREMENTS FOR A SORT (Cont.)

There is no positive optimum for facility assignment using the formula but an effective rule is: If LGTHWK is greater than 6ØØ,ØØØ, with T = 1, recompute with T = 3. Increase T to the maximum of 5, if necessary, until LGTHWK is less than or equal to 6ØØ,ØØØ. If LGTHWK is 6ØØ,ØØØ or less with T = 1, no intermediate tape units should be assigned. The sort can be accomplished in core and drum with one merge pass.

EXAMPLE 1:

Assume 1ØØ,ØØØ 2Ø-word items are to be sorted.

D = 2,ØØØ,ØØØ
I = 2Ø

With T = 1, LGTHWK = $2{,}ØØØ{,}ØØØ\left(\frac{21}{2Ø}\right) + 15{,}ØØØ = 2{,}115{,}ØØØ$

Result exceeds standards as well as the effective rule.

With T = 2, LGTHWK = $\frac{2{,}ØØØ{,}ØØØ\left(\frac{21}{2Ø}\right) + 15{,}ØØØ}{2} = 1{,}Ø57{,}5ØØ$

This result also exceeds standards and the effective rule.

With T = 4, LGTHWK = $\frac{2{,}ØØØ{,}ØØØ\left(\frac{21}{2Ø}\right) + 15{,}ØØØ}{4} = 528{,}75Ø$

This is an acceptable result, therefore

LGTHWK - 528,75Ø
Intermediate Tapes = 4
COREWK = $\frac{528{,}75Ø}{5Ø}$ = 1Ø,575 or 6 additional DBANK.

EXAMPLE 2:

Assume 5Ø,ØØØ 1Ø-word items to be sorted.

D = 5ØØ,ØØØ
I = 1Ø

With T = 1, LGTHWK = $5ØØ{,}ØØØ\left(\frac{11}{1Ø}\right) + 15{,}ØØØ = 565{,}ØØØ$

LGTHWK is less than 6ØØ,ØØØ with T = 1, therefore no intermediate tapes are required.

COREWK = $\frac{565{,}ØØØ}{5Ø}$ = 11,3ØØ or 6 additional DBANK.

13.4. INFORMATION RELATIVE TO THE STRAIGHT MERGE ONLY

A. Alternation if requested by the FAC card refers to the output.

B. It is not necessary to increment the drum table length tag, LGTHWK.

C. COREWK must be incremented to allow for double buffers for each input file.

1. It should equal $2F_1$, + $2F_2$, ... + $2F_N$

F_1 = largest block on file 1

F_2 = largest block on file 2

F_N = largest block on last file

D. Option coding, if used, can only be last pass.

E. When column 12 of the PMØ card is coded "2", a message "REQ NUMBER OF WAYS TO MERGE" is typed out awaiting a reply of one digit. The number of ways to merge (columns 48-49 of the PMØ card) is reduced to the number typed in which may range from 1 to the maximum number requested in the PMØ card. This feature removes the necessity of preparing numerous sets of schedule cards to merge n files of sorted data. The FAC and TAL cards should reflect the maximum facilities required. When a "one-way" merge is specified, input and output equipment types must be the same.

13.5. SIGNIFICANT CONSOLE MESSAGES

Typeout	
A. NN JRID CH C/U LABEL ERROR IMAGE rr mmddyy (3Ø chars. of label) LABEL rr mmddyy (3Ø chars. of label) Where NN = message number rr = reel number mmddyy = data	A. Y - Accept label N - The routine assumes that another tape has been placed on the unit and the label check is reinitiated.
B. .CCS.XXXXXXXXXXX .TCS.XXXXXXXXXXX Ccc Uuu LION ERR 23	B. A block checksum failure occurred and the tape in error is identified by channel and and unit.
C. SORT ERR 21 Ccc Uuu	C. Sequence error on final output. The input tape in error is identified by channel and unit.
D. CHECKSUM ERROR XXXXXXXXXXX INPUT CHECKSUM XXXXXXXXXXX OUTPUT CHECKSUM	D. This is an internal checksum failure. The sum of the data bits at the end of the first pass does not equal the sum of the data bits at the end of the last pass.
E. NN JRID .CCS.XXXXXXXXXXX.RC.XXXXX .TCS.XXXXXXXXXXX.RC.XXXXXX NN = Message Identifier	E. File checksum error on a IIIA tape (LION INTERNAL FORMAT). Y - Accept N - LION ERR 116
F. NN JRID .CCS.XXXXXX.RC.XXXXX .TCS.XXXXXX.RC.XXXXX NN = Message Identifier	F. File checksum error on a IIA tape (Census Compatible Format). Y - Accept N - LION ERR 1Ø9

13.6. NOTES ON SORT/MERGE

13.6.1. VARIABLE LENGTH ITEMS

It is possible to sort variable length items. The type of intermediate storage may be Uniservo IIIA or drum backed by IIIA's.

13.6.2. OUTPUT FILE LABEL

When both input and output are in LION INTERNAL FORMAT, free words (a maximum of 32), if present in the input tape label will be transferred to the output tape label.

13.6.3. SORT FILE FIXED LENGTH ITEM SIZE

By making an entry in columns 61-64 of the PMØ card, an item size different from the one specified in columns 24-29 can be sorted. If there is no first pass option coding and the new item size is smaller than the input record size, this change is automatic. When there is option coding or when the new record size is larger than the input record size, this change must be effected by first pass own coding setting the new item size into "ITMSIZ" before returning control to the SORT/MERGE. The automatic feature is not available for variable sized items. When columns 61-64 are coded to change the item size, columns 65-68 should be completed to specify the output record size.

13.6.4 FINAL OUTPUT FIXED LENGTH ITEM SIZE

By making an entry in columns 65-68 of the PMØ card an item size different from the one specified in columns 24-29 can be written on the final output tape. When the record size created is larger than the Sorted record, the record will be junk filled unless last pass option coding inserts data to complete the record.

13.6.5. INPUT/OUTPUT-IIA CENSUS COMPATIBLE FORMAT

When both input and output are IIA Census Compatible format, up to 9 tapes may be sorted on a one-for-one basis without rescheduling the SORT/MERGE program.

13.6.6. MULTI-FILE INPUT

It is possible to use a multi-file tape as input to the SORT/MERGE and have the following options: 1) Decimally specify the first or only file to sort; 2) Sort every file or only one file after reaching the starting point. The first tape mounted must contain the label block of the file to commence the sorting operation. The output is a single file. Card columns 52-53 of the PMØ card are a count of the total number of physical input tapes involved.

13.6.7. FILLING OUT REQUEST SHEETS

When filling out the "REQUEST FOR 11Ø7 PROGRAM PREPARATION SERVICES," you must indicate in the space labeled "ASSEMBLE" that a FORELF/ELF compiling sequence is desired.

13.6.8. ALLOCATION OF OPTION CODING

A special debugging aid is now available in the SORT. A word, located in octal location 34 of the IBANK of the SORT, contains the address of first pass coding in the left half and the address of last pass coding in the right half. If a sort terminates abnormally, last pass coding is not available during first pass, and first pass coding is not available during last pass. The option coding segments are stored on the drum and are brought into core at appropriate times.

* U. S. GOVERNMENT PRINTING OFFICE : 1971 431-996/1421

14. SERVICE PROGRAMS

SUMMARY OF SERVICE PROGRAMS

For further information on the following, refer to the Computer Systems Manual.

14.1. CENSER

Included in CENSER are the following options:

1) CT for card-to-tape in LION External Format*

2) CT for card-to-tape in LION Internal Format*

3) CD for card-to-drum in LION External Format*

4) TP for tape-to-print in LION External Format

5) IP for tape-to-print XS3 in LION Internal Format

6) IPF for Tape-to-print Fieldata in LION Internal Format

7) DP for drum-to-print in LION External Format

8) BP for bi-octal tape to print

9) BaLnxc for on-line printing of a IIA tape where a=board, n=loop, x=language (F for Fieldata, X for XS3), and c=spacing (1 for single, 2 for double)

*For all card operations a START card must be provided in addition to a STOP card with "))))))" in the first 6 columns followed by 3 blank cards.

The general form of the START card is:

Col. 1-6	Col. 7	Col. 13-36	Col. 37-42	Col. 43-48	Col. 73-78
/././.	f	DECK NAME	TRN JRID	TRN PARAM	REWIND

where "TRN PARAM" contains a transfer parameter to be used with the job transfer described above. It may be left blank, if no transfer is desired.

14.1. CENSER (Continued)

Transfers:

The transfer columns may be used to transfer a CENSER produced tape (or drum area) back to CENSER to perform another function. For instance, the START card in a card-to-tape operation may contain the JRID of "CENSER" in column 37-42 and the parameter "TP" in column 43-44. This will cause CENSER to print the card-to-tape reel upon completion of the card-to-tape operation with no operator intervention.

Differences in the START card for the three card operations:

CT (LION External Format):

Column 7 - should contain an asterisk "*"

Columns 13-24 - should contain the card file name, which becomes the tape file name.

CT (LION Internal Format):

Column 7 - should contain one of the following:

Z - fill with XS3 zeros, data is XS3

S - fill with XS3 spaces, data is XS3

I - fill with ignores, data is XS3

F - data is Fieldata, fill with spaces, fixed 20-word record, last 40 characters space fill.

Columns 13-36 - a four-word file name which becomes the tape label.

CD (LION External Drum Format):

Column 7 - should contain an asterisk "*"

Columns 13-24 - should contain the card file name, which becomes the drum file name.

Columns 37-42 - MUST contain JRID of transfer

Columns 43-48 - MUST contain PARAM for transfer

Further information may be found in the Census Computer Manual Part 12D-SV012.

14.2. CKSUM

This program reads a IIIA LION Internal Format tape, types out the label, the total number of items in the file, and the block count.

The programer must furnish the number of tapes and the largest block size if it exceeds 605 words. See CSM 12D-SV00.

14.3. SELTPR

This program searches a IIA or a IIIA tape for a predetermined block. When found, the remainder of the tape is printed or the target block is read and printed in bi-octal format until action is terminated by the programer.

The programer must supply the number of words to search for in every block and the contents of each word. He must also furnish information as to when to set the jump switch which terminates the run. He must also specify sequential or repetitive printing and the number of reels to be searched. All information will be typed in at the console. See CSM 12D-SV01.

14.4. AUTOCP

This program compares two tapes recorded in the same format. The differences are printed with block and word number indicated.

The programer must indicate options which will be typed in at the console:

	TYPEIN	
1st character - type of comparison	Ø	IIIA to IIIA
	1	IIIA to IIA
	2	IIA to IIA
2nd and 3rd characters - output mode	PT	paper tape
	MT	magnetic tape
4th character - input recording	0	variable
	1	IIA fixed (IIA tape only)
5th character - input language	X	XS3
	Ø	binary

If the largest block on tape exceeds 605 words, this difference must be noted. A TAL card must be used to increment the length tag LEN. See CSM 12D-SV02.

14.5. BTP

This program prints a limited number of blocks from a tape on the on-line printer. The format is bi-octal. Input may be either IIA or IIIA.

The programer must furnish the number of blocks to be printed. (Printing should not exceed 5000 lines.) CENSER BP will also achieve the same result. See CSM 12D-SV03.

14.6. TCOPY

This routine is used to duplicate a tape recorded in any format, fixed or variable block mode. All combinations of IIA and IIIA servos are possible. This is not recommended for IIA tapes.

The programer must indicate what types of tapes are to be used (e.g. IIIA to IIIA, etc.). See CSM 12D-SV04.

14.7. CTSIM

The input to this routine is a IIA tape prepared on the 1401 by a card-to-tape conversion. Z sentinels must be added to the card decks as follows: 6 beginning and 12 ending sentinel cards. Space or ignore fill should be used in the conversion. This routine is used when the card reader cannot be used on line for any reason. The output from this routine is in LION External Format and the BCX XS-3 characters are converted to Fieldata. See CSM 12D-SV06.

14.8. EDTCPY

The purpose of this routine is to convert and copy IIIA tapes, recorded in LION External Format, to IIA tapes. Data is converted from Fieldata to XS-3 and prepared for the off-line HSP.

If the programer desires to drop words other than words 20 and 21, a PMO card must be supplied specifying the exact characters to be dropped, separating digits by commas. Characters are numbered 1-132. Up to 12 digits may be specified.

After the last digit is supplied a period must be used on the PMO card.

Output of EDTCPY should be printed on the off-line HSP, using Board 3, Loop 4, single space. Begin printing on Line 2. (CSM 12D-SV08).

14.9. CARTAP

This program reads cards on-line and prepares a IIIA LION Internal Format tape in XS-3 language using a small deck of cards. It is useful for preparing test decks. Refer to CSM 12D-SV09.

14.9. CARTAP (Continued)

The programer must supply record lengths and number of data characters per card for the record size. The first card must be a label card as defined:

Column	Contents	Actual Record Length
1-6	/././.	
7-8	06	6
	08	8
	10	10
	12	12
	15	15
	20	20
	24	24
	30	30
	40	40
	60	60
	70	120

Column	Contents	No. Characters	Words/Record
9-10	36 (2 records per card)	36	6
	72	72	6
	48	48	8
	60	60	10
	72	72	12
	90	90	15
	30	30	15
	70	120	20
	60	60	20
	72	72	24
	36	36	24
	60	60	30

14.9. CARTAP (Continued)

Column	Contents	No. Characters	Words/Record
	30	30	30
	70	120	40
	60	60	40
	70	120	60
	60	60	60
	70	120	120
	60	60	120
11	⁄	fill character ignore	
	0	fill character XS3 zero	
	1	fill character space	
12	R	Rewind tape	
	⁄b	No Rewind	

Example of card which reads two cards, 60 characters from each to create a 20-word record:

/././. 2060 ⁄bR

Example of card which reads 1 card of 80 characters to create a 20-word record with ignore fill.

/././. 2070 ⁄bR

14.10. DATACO

This program will perform one or more of the following services upon a IIIA tape recorded in LION Internal Format:

a) correct or delete existing items in a file

b) insert new items

c) search for selected items and copy; list the target item.

d) drop files from a multi-file tape

e) change the internal file label

The programer must furnish a number of parameter cards. Details are too extensive to be mentioned here. Refer to the Computer Systems Manual Part 12D-SV010 for details.

14.11. COPIER

The purpose of this routine is to copy IIA or IIIA LION Internal Format tapes with or without conversion from XS3 to Fieldata or Fieldata to XS3. Options are entire item conversion, partial item conversion, or no conversion. Items must be fixed length. This routine is much faster than RUTT.
Refer to the Computer Systems Manual Part 12D-SV011 for details.

14.12. RUTT

The RUTT program will read a IIIA LION Internal Tape and either check read or copy the tape to a IIA tape(s) in Census compatible format for off-line printing.

	OPTIONS AVAILABLE ARE:	OPTION
14.12.1.	To check read a tape securing check sum and record count typeouts.	CH
14.12.2.	To copy fixed record tape(s).	IS
14.12.3.	To copy variable sized records.	VS
14.12.4.	To split each item into a number of output items (the resulting size of each item must be a factor of 120).	IS or VS plus PMO card

A specification sheet "VARIABLE PARAMETERS-PART 2" must contain the parameters required. Columns 20-22 of PMØ should contain 3 digits representing the number of items in the split. Column 23 should contain a period.

Example: 030.

RESTRICTIONS:

a) When the input item size is divided by the splitting factor and a remainder occurs, then the input item size is truncated by the remainder.

b) When the input item size is divided by the splitting factor and the quotient is not a factor of 120, then the RUTT will terminate.

c) All 3 digits to represent the number of items must be present if used in the split option. Use Ø to fill.

d) When the input block size for any option is greater than 1205, the table length tag "SIZA" must be increased in a TAL card. If the increment exceeds 200 words, the DBANK of the FAC card must be increased to two modules.

(Continued) OPTIONS AVAILABLE ARE: OPTION

14.12.5. To copy n number of blocks at the beginning of each tape. If the standard 100 blocks for testing is needed, specify RUTT100. — RUTT100

If the number of blocks is not 100, then a specification sheet "Variable Parameters Part 2" must contain the parameters required. Columns 12-15 of PMØ should contain a four-digit block limit followed by a period in column 16. If the period is not in column 16, then the entire tape will be transcribed. — IS plus PMO card

Example: 0010.

14.12.6. To bypass a number of blocks at the beginning of each tape and to copy the remainder, a specification sheet "Variable Parameters-Part 2" must contain the parameters required. Columns 12-15 of PMØ should contain the four-digit number of data blocks to be bypassed. Column 16 should contain a period. Column 17 should contain a minus. — IS PMO CARD

Example: 0010.-

14.12.7. If it is desired to combine all the data into one output file, then the options containing "S" may be changed to "H", which means hold the output with no rewind. The "S" option will create a tape for each segment or file.

Note: The information described should be included under "REMARKS" on both the "Request for Services" and the "Operating Instructions" submitted by the programer. If a specification sheet "Variable Parameter-Part 2" is required, it must be submitted with the request.

14.13. UTT

The UTT program will read a II tape(s) in Census compatible format and either check read or copy the tape to a IIIA LION Internal Tape.

	OPTIONS AVAILABLE ARE:	OPTION
14.13.1.	To check read a tape securing check sum and record count typeouts.	CH
14.13.2.	To copy fixed records of a certain size, supply the number of words per record as three digits.	nnn F nnn S nnn H
	The "F" means the output tape is rewound after this transcription. The "S" means the output file is closed after this transcription but the tape is not rewound. Thus many files may be placed on a tape(s). The "H" means the output file is still open after this transcription and the tape is not rewound. Thus many tapes may be combined into one file.	
14.13.3.	To combine two or more input records of a certain size nnn into a larger output item, a specification sheet "Variable Parameter-Part 2" must contain the following information. Columns 20-22 of PMØ should contain 3 digits representing the number of items to be combined into one output item. Column 23 must contain a period. If the combined output item size is greater than 600 words, the table length tag "SIZA" must be increased accordingly.	nnn F nnn S nnn H plus PMO Card and TAL Card if required

EXAMPLE: If thirty 40-word input items are to be combined as one output item, specify 040S option with PMØ columns 20-23 containing 030. and TAL card containing SIZA/600.

See option 2 for the meanings of F,S, and H.

Restrictions:

a) The input items must be adjacent.

b) Unless the total number of input items is a multiple of the specified combination factor, there will be a loss of items at the end of the file.

c) All 3 digits representing size of item or number of items must be present wherever used. Use Ø to fill.

14.13.3. Restrictions (Continued)

See option 2 for the meanings of F,S, and H.

Note: All necessary information should be included on both the "REQUEST FOR SERVICES" and "OPERATING INSTRUCTIONS." If any specification sheets are needed, they should be supplied with the request.

14.14. RUT702

The purpose of this program is to transcribe data from compatible tapes created by the IBM 1013/7702 transmission system, or on occasion by the UNIVAC 1004, to Uniservo IIA or IIIA tape units.

See the Computer System Manual Part 12D-SV013 for details.

14.15. UTT4C

This routine will transcribe data from either IIA or IIIA tapes onto IBM compatible tapes.

See the Computer System Manual Part 12D-SV014 for details.

14.16. MICRO

This program transcribes data recorded on a IIA or IIIA tape in BCD language at 556 c.p.i. density in the S/C 4411 Computer Printer tape format. In the next operation each IVC block will be recorded as a frame on microfilm.

See the Computer System Manual Part 12D-SV015 for details.

14.17. RUTA19

This routine accepts IIIA tapes recorded in LION Internal Format and reformats the data (with selective conversion) into 20-word lines on IIA tapes for display on the off-line printer. Each original word is displayed as two words, and each new record begins a new line.

Options available are:

a) no PM card - entire item conversion from binary to BCD-XS3

b) blank PMØ card - no conversion

c) PMØ card with selected punches beginning column 12 for word 1, column 13 for word 2, etc. on a one-for-one relationship --

1 means binary to XS3 conversion of a word

0 means octal to XS3 conversion

Ø means no conversion

14.17. RUTA19 (Continued)

Columns 12 - 77 of the PM0 card represent words 0 - 65 of the item. Columns 12 - 77 of the PM1 card represent words 66 - 131, etc.

If PM cards are needed, supply the information on the specification sheet "VARIABLE PARAMETERS - PART 2." See CSM 12D-SV016.

14.18. SALVAG

This program copies IIA (Census Compatible) or IIIA LION Internal Format tapes and skips blocks that cannot be read. The programer should supply a record size for IIA tapes and the block size for IIIA tapes.

See Computer Systems Manual Part 12D SV017 for further details.

14.19. 2 TO 3

This program will copy one or more IIA tapes in Census Compatible Format to a IIIA LION Internal Format tape.

See Computer Systems Manual Part 12D SV018 for more details.

14.20. 4C TO 4C

This program will copy a 4C tape to another 4C tape allowing a change of density.

See Computer Systems Manual Part 12D SV020 for further details.

14.21. TP4C

This program will print a 4C tape on the on-line high-speed printer (restricted to 5000 lines).

See Computer Systems Manual Part 12D-SV022 for further details.

14.22. SUTT1

This is a special version of the RUTT1 service routine that transcribes IIIA tapes in LION Internal Format to other IIIA tapes in the same format. It allows the creation of multi files; it allows the bypassing of files or blocks; it allows the copy of a specified number of blocks, it allows for the creation of one file from several, etc.

See Computer Systems Manual Part 12D-SV023 for specific details.

14.23. PSTRIP (PROGRAM SYMBOLIC TAPE STRIPPER)

This is a routine designed to allow the processing of a card deck for assembly which contains control statements not normally found in a program deck. These control statements enable you to retrieve lines of coding from previously assembled program elements, placing them into your program as desired.

See Part 12D SV019 of the Computer Systems Manual for details.

14.24. FOIST

This is a service program which renumbers FORTRAN source coding statements in an orderly and logical sequence in increments of 10 up to a maximum of 500 statement numbers per program element.

See Part 12 SV021 of the Computer Systems Manual for details.

14.25. SR2FOL

This is a general stepwise regression program.

See Part 14F of the Computer Systems Manual for a complete description.

14.26. MDMIC

This routine is a modified version of MICRO with no provision for user's own coding. See 12D-SV025 of the CSM.

14.27. RUTMDS

This routine transcribes IIA or IIIA tapes to a IVC tape in a format suitable for printing on the MOHAWK MDS7160. Original data must be in BCDXS3 language. See 12D-SV026 of CSM.

14.28. IMAGER-TWOPAS SYSTEM

This system of programs produces formatted tables suitable for off-line double pass printing. Provision is made for 132 lines a page, 185 characters per line, and 30 pages each of left and right pass printing. See 14D of the CSM.

14.26. EXERCISES

14.26.1. Whenever the need arises to convert a IIIA tape to a IIA tape, what service routine is available for our use?

14.26.2. The service routine RUTA19 is used for what purpose?

14.26.3. CENSER is the name of the Census Service System we use for on-line card-to-tape, tape-to-print, etc., functions. Which option is used for bi-octal printing?

14.26.4. What is the purpose of the service program PSTRIP?

14.26. ANSWERS

14.26.1. RUTT for off-line printing, or the faster routine COPIER. COPIER may be used depending upon the ultimate need.

14.26.2. To prepare a data tape for display printing with or without conversion of selected fields, 0 for octal, 1 for binary to excess - 3, or nothing for no conversion.

14.26.3. BP

14.26.4. To combine pieces of programs to create a new program.

15. DECISION TABLES

DECISION TABLES

A decision logic table is a table of all contingencies that are to be considered in the description of a problem, together with the actions to be taken. Decision tables are sometimes used in place of flowcharts for problem description and documentation.

Decision tables demand that all combinations or conditions be considered and allow irrelevant tests to be omitted. Once a program has been documented in decision table form, it is easy to update the documentation whenever changes in the program are required, because an individual table in the program can easily be changed.

15.1. LIMITED-ENTRY DECISION TABLES

In limited-entry form a decision table has four major sections as shown in the following diagram:

CONDITION STUB	CONDITION ENTRY
ACTION STUB	ACTION ENTRY

The condition stub in the upper left quadrant should contain (in question form) all those conditions being examined for a particular problem segment (for this table).

The action stub in the lower left quadrant should contain, in simple narrative form, all possible actions resulting from the conditions listed above.

The condition entry section in the upper right quadrant contains the answers to the question asked in the condition stub. Responses may be "Y" for yes and "N" for no. If no response is indicated it may be assumed that the condition was not tested in that particular combination.

15.1. LIMITED-ENTRY DECISION TABLES (Continued)

The action entry is in the lower right quadrant. The appropriate action resulting from the various combinations of responses to condition will be indicated here by an entry "X" which means to "take this action." A blank means "do not take the action shown."

The various combinations of responses to conditions shown in the condition entry portion of the table and their resulting actions are called rules or paths. Each column encompasses a rule. Each is given a number for identification purposes in the rule header portion of the table.

One other element each table requires is the table header or name to identify the table. A collection of tables may comprise the program.

The following outline shows all basic elements:

TABLE HEADER	RULE HEADER				
	1	2	3	4	-------
CONDITIONS					
ACTIONS					

EXAMPLE of Limited-Entry Decision Table

The standards for approving incoming credit orders are as follows:

If an order is for less than $700, it is to be automatically approved.

If an order is for more than $700, past records must be examined.

If the past credit record is bad, then the order must be referred to the manager who will decide whether or not to approve the order. Otherwise, the order will be approved.

The decision table for this example will look like the sample on next page.

15.1. LIMITED-ENTRY DECISION TABLES (Continued)

Sample:

LIMITED-ENTRY TABLE

CREDIT	1	2	3	4
IS CREDIT LIMIT OKAY?	Y	N	N	N
FAVORABLE PAYMENT RECORD?		Y	N	N
SPECIAL CLEARANCE OBTAINED?			Y	N
APPROVE ORDER	X	X	X	
DISAPPROVE ORDER				X

Notice that if testing of a condition is not necessary in a given situation, the rule entry space for that particular rule is left blank. In rule 1 on the credit table, if the credit limit is okay, the order may be approved without testing conditions 2 and 3.

15.2. EXTENDED-ENTRY DECISION TABLES

In extended entry tables the condition stub serves only to identify the variables to be tested, while the condition entry must define the value or state of the variable. This value may be absolute or related to another value. Likewise, in such a table, the action stub only names an action; the action entry will give the specifics for the action names. Use of the extended entry format tends to decrease the number of items listed in both the condition stub and the action stub, thus compressing the table vertically. It also offers the possibility of considering more than two responses to a given condition. However, it is often necessary to convert extended entry tables to limited entry form if they are to be programmed.

Sample:

EXTENDED-ENTRY TABLE

RACE/SEX	1	2	3	4
PERSON'S RACE?	CAU	CAU	OTHER	OTHER
PERSON'S SEX?	MALE	FEMALE	MALE	FEMALE
TALLY	A	B	C	D

Legend:

TALLY A - CAUCASIAN MALES

TALLY B - CAUCASIAN FEMALES

TALLY C - OTHER MALES

TALLY D - OTHER FEMALES

15.3. MIXED-ENTRY TABLES

When limited-entry form and extended-entry form are combined into a single table, the resulting table is said to be in mixed-entry form. Even though these two forms may be combined, one form must be used exclusively within each horizontal row of a table.

Sample:

MIXED-ENTRY TABLE

RACE/SEX	1	2	3	4
PERSON'S RACE?	CAU	CAU	OTHER	OTHER
PERSON'S SEX?	MALE	FEMALE	MALE	FEMALE
TALLY CAUCASIAN MALE	X			
TALLY CAUCASIAN FEMALE		X		
TALLY OTHER MALE			X	
TALLY OTHER FEMALE				X

15.4. DETERMINING MAXIMUM TABLE SIZE

In limited entry format each condition has one of two possible answers: "Yes" or "No." Based on this fact, the number of conditions in such a table dictates the maximum number of rules possible. The number of conditions in the table becomes the exponent of the number 2 to determine the possible number of rules. For example, a table having 3 conditions would have 2^3 (or 8) rules possible. Note that for each condition added to the table, its possible size doubles. A table with 4 conditions would have 2^4 (or 16) possible rules.

15.5. A METHOD OF ELIMINATING IRRELEVANT TESTS

A table will require the maximum number of rules only when no logical interaction among the various conditions is involved. However, when logical interaction is involved, the table will not use the maximum number of rules possible.

If two rules result in the same action (or actions) and there is only one difference in their responses, the condition for which the responses are different should not be tested.

Example:

The following decision table lists all the rules for issuing cruise tickets (cabin or tourist class) on a steamship:

CRUISE (ALL POSSIBLE RULES)	1	2	3	4	5	6	7	8
IS REQUEST FOR CABIN CLASS?	Y	Y	Y	Y	N	N	N	N
IS CABIN AVAILABLE?	Y	Y	N	N	Y	Y	N	N
TOURIST AVAILABLE?	Y	N	Y	N	Y	N	Y	N
ISSUE CABIN TICKET, SUB 1 FROM TOTAL	X	X						
PLACE ON CABIN WAITING LIST			X	X				
ISSUE TOURIST TICKET, SUB 1 FROM TOTAL					X		X	
PLACE ON TOURIST WAITING LIST						X		X

15.5. A METHOD OF ELIMINATING IRRELEVANT TESTS (Continued)

Notice that for rules 1 and 2 the same action is taken regardless of the last condition test. Therefore, rules 1 and 2 can be combined into one rule and not making the test for "tourist available." Also notice that for rules 3 and 4, they may be combined into one rule and omitting the test for "tourist available." Note that for rules 5 and 7 the test for "cabin available" may be omitted and then only one rule is required. Also notice that rules 6 and 8 may likewise be combined into one rule. The decision table should then be simplified to look like this:

CRUISE (NECESSARY RULES)	1	2	3	4
IS REQUEST FOR CABIN CLASS?	Y	Y	N	N
IS CABIN AVAILABLE?	Y	N		
TOURIST AVAILABLE?			Y	N
ISSUE CABIN TICKET, SUB 1	X			
PLACE ON CABIN WAITING LIST		X		
ISSUE TOURIST TICKET, SUB 1			X	
PLACE ON TOURIST WAITING LIST				X

15.6. BIFURCATED TABLE ARRANGEMENT

A table may be rearranged in any fashion as long as you do not change the responses ("yes" or "no") to any condition within a rule and as long as the path for each rule continues to result in the same action.

In a bifurcated table the first condition must have all "yes" responses grouped together and all "no" responses grouped together, with no blanks between the first and last rule of the condition entry. It is allowable to have the "no" responses first. Each following condition must have similar groupings. Since blanks are not permitted in a path, the sequence of the rules and conditions must be changed. The condition having the greatest number of responses must appear first in the table. Then the condition having the second greatest number of responses, etc. After this is done, the rules must be rearranged. There are usually several valid arrangements.

An example of a bifurcated table follows (before and after).

TABLE BEFORE BIFURCATION	1	2	3	4	5
CONDITION 1	Y	Y	Y	N	Y
CONDITION 2	N	N	N		Y
CONDITION 3		Y	N		
CONDITION 4	Y	N	N		
ACTION 1			X	X	
ACTION 2					X
ACTION 3	X				
ACTION 4		X			

TABLE AFTER BIFURCATION	1	2	3	5	4
CONDITION 1	Y	Y	Y	Y	N
CONDITION 2	N	N	N	Y	
CONDITION 4	Y	N	N		
CONDITION 3		Y	N		
ACTION 1			X		X
ACTION 2				X	
ACTION 3	X				
ACTION 4		X			

15.7. THE ELSE RULE

If a table cannot be arranged in bifurcated form, chances are that it contains contradicitions, redundancies, or omissions. In a truly bifurcated table containing 2^n rules, the rules in combination are exhaustive and mutually exclusive. An important exception to this is a table containing an ELSE rule. The ELSE rule, by definition, includes all rules not specifically covered in the table. Usually the rule is nothing more than a rule column with blanks shown as response to all conditions and an appropriate action indicated by an "X". The ELSE rule may be used to detect "illegal" codes.

15.8. SEQUENCE CONTROL DEFINITIONS

Sometimes decision tables contain such statements as START, RETURN, EXIT, and GO TO -. When coding for a preprocessor these terms must be expressed in a suitable manner to meet the requirements of the language. However, a definition of these terms is included merely for information.

START - This condition entry indicates action to be taken on each reference to the table from outside.

RETURN - The action entry which sends control reference to the first rule and condition in the entry table.

EXIT - The action entry which serves as an end point for a table that is referenced by means of a DO statement. EXIT contains a transfer to the table line following the DO statement. This allows one table to reference another as a subroutine.

GO TO - This action entry means control will be transferred.

15.9. EXAMPLES OF DECISION TABLES

Problem 1: Given an array X of size N, determine whether or not X is in sequence, storing the result in the logical variable ANS.

DECISION LOGIC TABLE:

START	Y	N	N	N
$I \geq N$		Y	N	N
$X(I) \leq X(I+1)$			Y	N
I = 0	X			
I = I+1	X		X	
RETURN	X		X	
ANS = "YES"		X		
ANS = "NO"				X
EXIT		X		X

FORTRAN CODING:

(a)
```
      I = 0
   10 I = I + 1
      IF (I.GE.N) GO TO 20
      IF (X(I).LE.X(I+1)) GO TO 10
      ANS = .FALSE.
      GO TO ...
   20 ANS = .TRUE.
      GO TO ...
```

(b)
```
      DO 10  I = 2,N
      IF (X(I-1).GT.X(I)) GO TO 20
   10 CONTINUE
      ANS = .TRUE.
      GO TO ...
   20 ANS = .FALSE.
      GO TO ...
```

The coding (a) represents a direct translation of the decision table, whereas (b) is less direct but more efficient in terms of execution time.

15.9. EXAMPLES OF DECISION TABLES (Continued)

Problem 2: Given an array X of size N, find its maximum and minimum values.

DECISION LOGIC TABLE:

START	Y	N	N	N	N
I > N		Y	N	N	N
X(I) ≧ MAX			Y	N	N
X(I) ≦ MIN				Y	N
I = 1	X				
MAX = X(I)	X		X		
MIN = X(I)	X			X	
I = I+1	X		X	X	X
RETURN	X		X	X	X
EXIT		X			

FORTRAN CODING:

(a)
```
      I = 1
      MAX = X(I)
10    MIN = X(I)
20    I = I + 1
      IF (I.GT.N) GO TO ...
      IF (X(I).GE.MAX) GO TO 30
      IF (X(I).LE.MIN) GO TO 10
      GO TO 20
30    MAX = X(I)
      GO TO 20
```

(b)
```
      MAX = X (1)
      MIN = X(1)
      DO  10  I = 2,N
      IF (X(I).GT.MAX)  MAX = X(I)
      IF (X(I).LT.MIN)  MIN = X(I)
10    CONTINUE
```

The coding (b) is actually a mistranslation of the decision table, but is more efficient with respect to running time and number of instructions.

15.9. EXAMPLES OF DECISION TABLES (Continued)

Problem 3: Given a positive integer N, determine the number of decimal digits NOD contained in N.

Examples:

N	36	7	1965	100	586437
NOD	2	1	4	3	6

DECISION LOGIC TABLE:

START	Y	N	N
N<T		Y	N
NOD = 1	X		
T = 10	X		
NOD = NOD+1			X
T = T*10			X
RETURN	X		X
EXIT		X	

FORTRAN CODING:

```
      FUNCTION NOD(N)
      INTEGER T
      NOD = 1
      T = 10
   10 IF (N.LT.T) RETURN
      NOD = NOD + 1
      T = T *10
      GO TO 10
      END
```

15.9. EXAMPLES OF DECISION TABLES (Continued)

Problem 4: Given positive integers A and B, determine if A is evenly) divisible by B.

Examples: A	12	19	7	30	15	6
B	4	6	7	12	1	24
Is A divisible by B?	YES	NO	YES	NO	YES	NO

DECISION LOGIC TABLE:

A:B	<	=	>
A = A - B			X
RETURN			X
DIV = "YES"		X	
DIV = "NO"	X		
EXIT	X	X	

FORTRAN CODING:

```
      LOGICAL FUNCTION DIV(A,B)
      INTEGER A,B
10    IF (A-B)  40,30,20
20    A = A-B
      GO TO 10
30    DIV = .TRUE.
      RETURN
40    DIV = .FALSE.
      END
```

15.9. EXAMPLES OF DECISION TABLES (Continued)

Problem 5: Given an NxN matrix A, obtain its transpose without using an auxiliary array.

Note: A "matrix" can be most conveniently represented by a two-dimensional array. The "transpose" of a matrix is the matrix obtained by interchanging rows and columns. For example, the two matrices below are transposes of each other.

1	2	3
4	5	6
7	8	9

1	4	7
2	5	8
3	6	9

DECISION LOGIC TABLE: 5A

(a)				
	START	Y	N	N
	$I \geq N$		Y	N
	I = 0	X		
	DO 5B			X
	I = I+1	X		X
	RETURN	X		X
	EXIT		X	

FORTRAN CODING:

```
(a)         I = 0
        10  I = I + 1
            IF (I.GE.N)  GO TO ...
```

DECISION LOGIC TABLE: 5B

(b)				
	START	Y	N	N
	J > N		Y	N
	TEMP=A(I,J)			X
	A(I,J)=A(J,I)			X
	A(J,I)=TEMP			X
	J=I	X		
	J=J+1	X		X
	RETURN	X		X
	EXIT		X	

FORTRAN CODING:

```
(b)         J = I
        20  J = J + 1
            IF (J.GT.N) GO TO 10
            TEMP = A(I,J)
            A(I,J) = A(J,I)
            A(J,I) = TEMP
            GO TO 20
```

15.10. MACHINE PROCESSING OF DECISION TABLES

Although many programers still code by hand directly from completed tables, computer processing of the tables is now a reality.

The programer codes as usual those segments of the program not covered by decision logic tables. He indicates in his coding which tables are to be inserted and exactly where in the program they are to be placed. The tables will result in additional coding being generated and inserted into the program.

At the Bureau of the Census, a decision table preprocessor is available for use with both FORTRAN and COBOL. The name of this program is TAB4O. The Census original version is referred to as TAB7C for use on the Univac 1107, and that version was later modified and called TAB4IX. A description of TAB4IX follows.

15.11. TAB4IX: A DECISION TABLE PREPROCESSOR FOR THE UNIVAC 1107 FORTRAN IV COMPILER

This version of the decision table preprocessor was originally written by Mr. Lewis T. Reinwald while he was with RAC, for use on the IBM 7Ø4Ø. The program was originally known as TAB4Ø. Different versions of the program were converted by members of the Computer Sciences Staff for use on the 1107. The current version is known as TAB4IX.

On facility AT the decision table preprocessor, TAB4IX, accepts a card-to-tape input deck made up of TAB4IX control cards, standard FORTRAN IV statements and decision tables formatted for TAB4IX. There are three outputs: One is a PNCHT tape which can be used as input to an original FORTRAN assembly; another is a PRNTT tape which produces a listing of the input with line numbers (for corrections) and error warnings, and the third which is on facility CT can be used as input to TAB4IX on facility AT in a correction pass if required. The card-to-tape correction deck is on facility BT. The format for the card-to-tape reel on BT is described in 15.11.9.

Essentially, the preprocessor builds FORTRAN statement fragments into complete FORTRAN statements and keeps track of label references.

A look at a simple example might be more useful than an attempt to list formally all of the features of the decision table preprocessor.

15.11. TAB4IX: A DECISION TABLE PREPROCESSOR FOR THE UNIVAC 1107 FORTRAN IV COMPILER (Continued)

DECISION LOGIC TABLE

		1	2	3
1	AIYP > ≤ OWN	Y	Y	N
2	OLRO = Ø	Y	N	
1	AIYP - ≤ OWN → LOWN		X	
2	AIYP + LRO → LRI	X		
3	LRO + (≤ OWN-AIYP) → LRO			X
4	GO to Table 2-4	X	X	X

Table above formatted as input to TAB4IX

```
.2Ø1Ø  CONTINUE
C**    TABLE 2-1Ø
2  B                          R     R     R
T  AIYP.GT.ISWN               .T    .T    .F
T  OLRO .EQ.Ø                 .T    .F    ..
A  LOWN = AIYP - ISWN         ..    .X    ..
A  LRI = AIYP + LRO           .X    ..    ..
A  LRO = LRO + ISWN           ..    ..    .X
       - AIYP
A  GO TO 2ØØ4                 .X    .X    .X
.2Ø11 CONTINUE
```

15.11.1. The second line shown has a "C" in column 1 and indicates that what follows is a comment. A comment in the FORTRAN part of a program will appear on the PNCHT and PRNTT tapes, but a comment in the table part of the program will appear only on the PRNTT tape. The TAB4IX program "expects" the first card to be a FORTRAN statement if it doesn't have a "table number" in column 1 as described in 15.11.2.A. In this case, the comment would be treated as a FORTRAN comment.

15.11.2. The first line of the table (third line shown) formatted as input to TAB4IX which reads

"2 B R R R ."

can be explained as follows:

A. Card Column 1

1. The "2" in card column 1, known as the table number, indicates to the preprocessor that this card is the beginning of a table as opposed to a line of FORTRAN coding or a TAB4IX control card. No FORTRAN statement in a program to be preprocessed by TAB4IX should have a statement number of five digits, as these statement numbers would be interpreted as table numbers by the preprocessor.

2. Instead of a "2" here, we could have used a "1" or a "3" (See 15.11.7.B. for the limitation on number of rules with a table number of 3). The table number need not be unique within a program. A table started with a "Ø" in card column 1 has a special use explained in 15.11.13.A.

B. The "B" in card column 3 indicates to TAB4IX that the columns between the "B" and the next occurrence on this line (here an "R") are to be used as the beginning part of any FORTRAN statements formatted in this table. It is important to notice that when fragments are combined, they should form legal FORTRAN statements. This explains the use of ".GT." and ".EQ." in T (test) clauses and "=", "+" and "-" in A(action) clauses. The entries under the B are the result of "factoring out" from the beginning parts of the FORTRAN statements, those parts which are common to a line or clause in this table.

C. "R" - same card as above. The "R" stands for the rules of a decision table. Only two columns are required for each of the rules in this table; room enough for ".T", ".F", ".X" or "..". The "." after the third R is an indication to the preprocessor that it need not scan the card for data beyond the ".". The T or A must not appear on the same margin as the R. However, the table shown could be reversed, with the R on the vertical margin and T and A on the horizontal margin.

15.11.3. The first and last lines shown which read ".2Ø1Ø CONTINUE" and ".2Ø11 CONTINUE" are used as FORTRAN addressable entrances to tables.

A. The "2Ø1Ø" line is an entrance to the table shown. The "2011" line is an entrance to the table, or coding, following the table shown.

B. The "." in column one for these "CONTINUE" lines is necessary as this is the indication to the preprocessor of the first FORTRAN line following a table.

15.11.4. The two lines which read "T AIYP .GT. ISWN .T .T .F" and "T OLRO.EQ.Ø .T .F .." are made up of FORTRAN fragments (the parts occurring under the "B") and implicit entries (".T", ".F" and ".."; the parts occurring under the R's).

A. "T" in column 1 indicates to the preprocessor that it is dealing with a test and in this case is concerned with FORTRAN lines of coding which when assembled could read:

"IF(.NOT.(AIYP.GT.ISWN)) GO TO 2Ø338"

or "IF(.NOT.(OLRO.EQ.Ø)) GO TO 2Ø236".

B. The ".T" under the R's corresponds to the "Y's" or "Yes path" in the decision tables; the ".F" corresponds to the "N's" and ".." is an indication to the preprocessor to ignore a cell. The ".T" and ".F" have meaning only for TESTS (T); the ignore (..) is applicable to the A(action) also.

15.11.5. The four lines which begin with "A" in column 1 are written in FORTRAN format and are "actions" to be taken if the above rules are "satisfied." The ".X" under the second rule for the first A line indicates that variable LOWN is set equal to AIYP - ISWN. The ".." in the first line, but under the first and third rules indicates that this action is to be ignored for these rules. It is worth noting that a statement (T or A) can be continued on another line, as on the third A line. The .X has no meaning for a "T" line.

15.11.6. EXPLICIT ENTRIES

The decision table could be rewritten without the use of the B described in 15.11.2.B. Here the FORTRAN fragments appear under the rules. In this table, the entries "AIYP.GT.ISWN" and "GO TO 2Ø44" are called explicit entries. The implicit entries (.T,.F,..,.X) refer to the nearest explicit entry on the left.

2	R	R	R
T	AIYP .GT. ISWN	.T	.F
T	OLRO .EQ.Ø	.F	..
A	..	LOWN = AIYP - ISWN	..
A	LRI = AIYP+ LRO	..	..
A	..	..	LRO = LRO + ISWN- AIYP
A	GO TO 2ØØ4	.X	.X

15.11.7. L - Card.

The following example illustrates the use of the L - Card.

		1	2	3
1	AIYP>ΣOWN	Y	Y	N
2	OLRO = Ø	Y	N	
1	ΣOWN =ΣOWN - 1		X	
2	ΣOWN =ΣOWN - 2	X		
3	ΣOWN =ΣOWN - 3			X
4	GO to Test 1, Rule 1			X
5	GO to Table 2 - 2		X	
6	GO to Table 2 - 3	X		

Table above formatted as input to TAB4IX

2	B	R	R	R
L		2Ø1		
T	AIYP.GT.IOWN	.T	.T	.F
T	OLRO.EQ.Ø	.T	.F	..
A	IOWN = IOWN -	2	1	3
A	GO TO 2	ØØ3	ØØ2	Ø1

A. The L-card assigns statement number 2Ø1 to the first rule and first test of the table.

B. FORTRAN FORMAT statements must always be assigned statement numbers with an L-card. If a test and a rule are referenced within a table by a "GO TO," assign an "L-card" to the particular rule as in the example. The statement number assigned should be less than five digits to avoid conflict with the five digit statement numbers assigned by TAB4IX. If any tables use table number "3", there should be less than 27 rules in all.

15.11.8. TAB4IX CONTROL STATEMENTS

A. There are five TAB4IX control statements used for the FORTRAN preprocessor and they all have a "-" punch in column 1.

B. Two of the control statements are the standard FOR and MAP cards and except for the "-" in column 1, instead of the 7/8 punch, they function as in FORTRAN for the UNIVAC 1107.

C. A "-IBSYS" card should be the last card before the Standard Stop card on any input to TAB4IX.

D. Some coding which is required at several places in the program, such as COMMON statements, can be stored initially and called as needed. This is done by the "-TASAV" and "-TARTV" statements.

The first card used is the -TASAV card with a name starting in column 16. This name should have six or less characters. The cards that follow contain the images to be stored. A "-TASAV" card with blanks in columns 16-21 ends the deck to be stored. A "-TARTV" card with the same name in columns 16-21 as the "-TASAV" card used, will enter these images into the program at the place the "-TARTV" card appears.

15.11.9. TAB4IX SETUP

The input to TAB4IX (facility AT) is a card-to-tape output, or CT output from TAB4IX. The PNCHT output from TAB4IX can be used as input to an original FORTRAN assembly; the PRNTT facility output is to be printed.

15.11.10. CORRECTION PASS

A card-to-tape output which is used as input to TAB4IX, facility AT, might need to be changed. The PRNTT tape from TAB4IX shows card numbers (starting in column 74). A "+" in column 1 can be used as a control punch in the same way as "-" is used in making standard FORTRAN corrections.

Insertion

"+1" means insert after card number 1, the following corrections.

Replacement

"+1,1" means delete card number 1, replace with following corrections.

Deletion

"+1,1" means delete card number 1.

The corrections are preceded and followed by standard start and stop cards; the deck is card-to-taped and used as input on facility BT; the output from the prior run, facility CT, is entered on facility AT.

15.11.11. PRNTT TAPE

In addition to the listing of the input to TAB4IX with the card numbers, the PRNTT tape will show some diagnostics for the decision tables. If a table is incomplete (no ".F's" could make a table incomplete) TAB4IX will generate a "CALL DUMP." This will result in a compiling error unless the input program is provided with a subroutine named DUMP that initiates a core dump operation. (See 15.11.14.B. for example of an incomplete table.)

ELSE Rule - An Else rule does not contain any condition entries. The Else rule, when used, appears as the last rule in the table. If none of the rules on the left of the Else rule have been satisfied, then control is directed to the actions shown in the Else rule. When this condition arises and there is no else rule provided, the preprocessor will generate "CALL DUMP" to indicate an incomplete table as discussed in 15.11.11.

15.11.12. BOUNDS

A. Number of input cards.
No table should have more than 3ØØ cards as input.

B. Number of =TASAV input cards.
The sum of the number of cards read in as "-TASAV" cards should be less than 2Ø1.

C. There should be less than eleven sets of -TASAV cards.

D. Punched cards prepared for input to TAB4IX should not use more than 72 columns.

E. TAB4IX assigns statement numbers to generated FORTRAN statements by taking the table number as the first digit, the rule number as the next two digits and the clause number as the last two digits. In 15.11.6. the line which reads "LRI=AIYP+LRO" is clause "Ø4". The line which reads "OLRO .EQ. Ø is clause "Ø2". Clause numbers are numbered consecutively and continuously for each table number. When more than 99 clauses have been encountered, the preprocessor will increase the rule count by the maximum number of rules encountered (plus one) in any previous decision table having the same table number as the current table. The current table and the previous tables must be part of the same sub program. When the number of rules exceeds 99, the preprocessor lists a warning on the PRNTT tape of the possible existence of duplicate statement numbers. By alternating the table numbers used, duplication should be avoided. When a "FOR" card is encountered, statement numbers start over for each table number. The only statement numbers shown are those which are referenced.

15.11.13. There are additional codes that can be used on the margins of decision tables. Their use should be limited. The main virtue of the preprocessor is that decision table specifications can be quickly copied in order to get a running program that accurately reflects the job specifications. The use of some of these additional codes can lead to some complicated constructions that detract from the simplicity of TAB4IX. Examples for A-G are shown in paragraph 13.

A. Table number Ø. Table number Ø differs from tables numbered with 1, 2 or 3 mainly in two ways:

(1) No FORTRAN IF statements are generated, therefore no conditions are involved.

(2) The T for test is omitted. There is an "R-margin" and an "A-margin".

B. E - An "E" is used much as a "B", but the E stands for End. Those fragments which are common to the "End" of a clause are "factored out" and written under an "E".

C. K - The K concatenates a fragment to the end of the rule entry.

D. "*" - For an action, the "*" is the same as a "K". For a test, an "*" concatenates with an ".AND." connector.

E. "/" = Slash is the same as "*" except "/" concatenates with an ".OR." for a test.

F. S - or Statement. The "S" appears on the T - margin and indicates an action to be taken regardless of the outcome of the tests.

G. P - Procedure. Differs from a rule in that it is considered even if rule on its left is satisfied. See the difference between the coding generated, for examples 15.11.14.B. and 15.11.14.C.

15.11.14. EXAMPLES FOR PARAGRAPH 15.11.13.

A.

```
Ø   B              E    A     A     A     A
R   EQUIVALENCE(        LR    LO    IR    JO
K   ,              )    LCL   LPH   ICL   IPH
```

The above would generate these FORTRAN statements:

```
      EQUIVALENCE(LR,LCL)

      EQUIVALENCE(LO,LPH)
      EQUIVALENCE(IR,ICL)
      EQUIVALENCE(JO,IPH)
```

B.

```
1   B              E    R     R
S   X = X +             1.    2.
T   X.EQ.               1.    2.
/   Y.EQ.               2.    3.
A   X = X +             1     3
K   4              5    2     6
```

The above could generate:

```
      X = X + 1.
      IF(.NOT.(X.EQ.1..OR.Y.EQ.2))GO TO 1Ø2Ø1
      X = X + 1425
      GO TO 1Ø3Ø4
1Ø2Ø1 X = X + 2.
      IF(.NOT.(X.EQ.2..OR.Y.EQ.3.))CALL DUMP
      X = X + 3465
      GO TO 1Ø3Ø4
1Ø3Ø4 CONTINUE
```

15.11.14. EXAMPLES FOR PARAGRAPH 15.11.13. (continued)

C.

```
      1  B        E    R    P
      S  X = X +         1.   2.
      T  X.EQ.           1.   2.
      /  Y.EQ.           2.   3.
      A  X = X +         1    3
      K  4        5      2    6


      X = X + 1
      IF(.NOT.(X.EQ.1..OR.Y.EQ.2.))GO TO 1Ø2Ø6
      X = X + 1425
      GO TO 1Ø2Ø6
1Ø2Ø6  X = X + 2.
      IF(.NOT.(X.EQ.2..OR.Y.EQ.3.))CALL DUMP
      X = 3465
      GO TO 1Ø3Ø9
1Ø3Ø9  CONTINUE
```

15.11.15. COMPARISONS BETWEEN "STRAIGHT FORTRAN" CODING AND THE TAB4IX PREPROCESSOR

A. Programing and debug time

At this date there is not too much direct comparison between these two forms of programing, but there should be considerable saving (50%) in using TAB4IX when the specifications are in decision table format.

B. Preprocessor time

The TAB4IX decision table preprocessor can be written in about 15ØØ lines (cards) of decision table format. It takes about three minutes and 40 seconds to preprocess this deck. At the present time this is the best estimate we have for preprocessor time per number of cards.

C. Core Space

At this time it seems that a program which is written in decision table format, processed through TAB4IX and then compiled, will take twenty to thirty percent more room in core than one which is coded from decision tables in straight FORTRAN.

D. Running Time

The running time of the compiled program output from TAB4IX seems to be comparable to that of one coded in straight FORTRAN.

15.11.16. 1107 OPERATING INSTRUCTIONS FOR TAB4IX

FORM BC-558 (3-17-67)

U.S. DEPARTMENT OF COMMERCE
BUREAU OF THE CENSUS

1107 OPERATING INSTRUCTIONS

A. Div. code	B. Project number	C. Program name	D. Sponsor's name	E. Tel. Ext.	F. Job number (PD)
		TAB4IX			

G. Program information

1. [XX] TEST *(Other than sort/merge)*

2. TRANSCRIPTION
   - [ ] PRIOR TO TEST
     - [ ] UTT
     - [ ] RUTT
   - [ ] AFTER TEST

   APPROXIMATE NUMBER OF OUTPUTS

3. [ ] SORT [ ] MERGE
   a. OPTION CODING USED [ ] OWN [ ] DUMMY [ ] NONE
   b. EXTERNAL LIBRARY NAME
   c. PROGRAM TO BE USED *(If other than standard)*

4. LIBRARY MAINTENANCE
   a. TYPE OF SUBROUTINE [ ] LOAD TIME [ ] ASSEMBLY TIME
   b. NO. OF COPIES REQUIRED
   c. EXTERNAL LIBRARY NAME

TO BE COMPLETED BY PROCESSING DIVISION

a. BINARY TAPE NO.	a. OWN CODING TAPE NO.	a. OWN CODING TAPE NO.
b. SOURCE [ ] FURNISHED [ ] FROM ASSEMBLY	b. SOURCE [ ] FURNISHED [ ] FROM LIBRARY MAINTENANCE	b. SOURCE [ ] FURNISHED [ ] FROM ASSEMBLY

H. Magnetic tape input(s)

[ ] NONE
[XX] INDICATE

(If servo alternation specify in "M")

TAPE NUMBER (a)	TRANS. TAPE NO. (DPSD) (b)	ASSIGNED FACILITY NAME (c)	EXTERNAL LABEL (d)	REMARKS (e)
		AT		Original card-to-tape or "CT" output
		BT		Corrections (if any)
				from card-to-tape

I. Magnetic tape output(s)

[ ] NONE
[XX] INDICATE

(If servo alternation specify in "M")

ASSIGNED FACILITY NAME (a)	EXTERNAL LABEL (b)	TAPE NUMBER (DPSD) (c)	TRANS. TAPE NO. (DPSD) (d)	REMARKS (e)
CT				To be used as CT input in next TAB4IX run
PNCNT				To FORTRAN original assembly
PRNTT				PRINT TAPE

J. Jump switches

NAME	CONDITIONS FOR SET AND/OR RELEASE

16. PROGRAMING

TECHNIQUES AND TIPS

16.1. FORTRAN: EXTERNALLY EXPANDABLE ARRAYS USING NAMED COMMON

The dimensions of Named COMMON areas in FORTRAN programs may be expanded at run time (by the use of schedule TAL cards) without a recompilation step. This technique is useful in general purpose programs where the dimensions of certain areas depend on varying externally controlled conditions.

Example 1:

If the statement:

```
COMMON/NAME/JARAY (100)
```

appears in the program, and a schedule TAL card is used which appears as follows:

```
JRID, TAL, LNAME/300.
```

then the true dimension of JARAY is (400).

Note that "NAME" and "LNAME" together with the initial length of the table will all appear under the heading "CORE" in the Elf Facility Synopsis.

Example 2:

If the statement:

```
COMMON/NAME2/KARAY (2,5,50)
```

appears in the program, and a schedule TAL card is used which appears as follows:

```
JRID, TAL, LNAME2/10.
```

then the true dimensions of KARAY are (2,5,51)

Rule: In a multi-dimensioned array of N dimensions, only the Nth dimension is affected by the increase shown on the TAL card. For this reason, the increase must be a multiple of the product of the first N-1 dimensions.

In the above example, any increase in KARAY must be in multiples of 10 (the product of the first two dimensions).

When using the TAL card to increase a named COMMON area, the letter "L" must precede the name of the area.

16.1. FORTRAN: EXTERNALLY EXPANDABLE ARRAYS USING NAMED COMMON (Cont.)

Example 3:

If the statement:

COMMON/NAME3/LARAY (50,10)

appears in the program, and a schedule TAL card is used which appears as follows:

JRID, TAL, LNAME3/500.

then the true dimensions of LARAY are (50,20).

Example 4:

If the statement:

COMMON/NAME4/MARAY (50)/NAME5/NARAY (80)

appears in the program, and a schedule TAL card is used which appears as follows:

JRID, TAL, LNAME4/40, LNAME5/50

then the true dimension of MARAY is (90) and the true dimension of NARAY is (130).

Example 5:

If the statement:

COMMON/NAME6/JARAY (50,10), KARAY (30)

appears in the program, and a schedule TAL card is used which appears as follows:

JRID, TAL, LNAME6/60.

then the true dimensions of JARAY remain at (50,10) and the true dimension of KARAY IS (90).

Note that in the case where more than one array appears in the same COMMON area, only the last array is affected by the increase shown on the TAL card.

16.2. COMPILING LARGE FORTRAN PROGRAMS

During the compilation of very large FORTRAN programs, there is a danger of overflowing the capacity of the FORTRAN compiler. However, if the program is well "segmented" into a series of external subroutines (each external subroutine is handled separately by the compiler), there is actually no practical limit to the size of FORTRAN programs. Large programs should be designed with this in mind from the beginning. Generally, each external subroutine, as well as the main program itself, should be limited to less than 700 FORTRAN statements each.

A specific problem has been encountered with programs containing a large number of DATA statements in the body of the main program. To avoid this problem, the DATA statements may be compiled separately from the main program in one or more external subroutines. By the use of COMMON statements, the DATA statement contents are made available to the main program or to any other external subroutine in the program.

A simplified example showing the use of COMMON and DATA statements in an external subroutine is shown below:

```
/./*      ANYPRO                                    REWINT

@ MAP ANYPRO,P.

@LIP FOR ANYPRO,ANYPRO,P.
     COMMON/NAME/NARRAY(10)
          .
          .
          .
     CALL SUB     (first executable statement)
          .
          .
          .
     END

@LIP FOR SUB,SUB,Q.
     SUBROUTINE SUB
     COMMON/NAME/NARRAY(10)
          .
          .
          .
     DATA (NARRAY(J),J=1,10)/1,2,3,4,5,6,7,8,9,10/
          .
          .
          .
     RETURN
     END
)))))
```

Note the one-time CALL to the data subroutine. This is necessary to cause the data subroutine to be allocated.

For a description of "SEGFOR", the program which provides for drum overlay segmentation, see Part 16.17.

16.3. DEFINING SUBROUTINE FORMAL ARGUMENTS

The following examples suggest a method for defining formal arguments to minimize code generation and execution time.

When formal arguments are referenced in the body of a subroutine, each reference is posted with the address of its corresponding actual argument in the subroutine call statement.

It therefore becomes desirable to minimize the number of references to the formal arguments of the subroutine. This can be accomplished by assigning each formal argument to a variable where possible, and then referencing the variable in the body of the subroutine instead of the formal argument.

Example:

Usual Method

```
      CALL PATCH (I1,I2,I3,I4)
      ...
      SUBROUTINE PATCH (J1,J2,J3,J4)
      DIMENSION J1(J2),J3(J4)
      ...
      IF(N.EQ.J2) GO TO 5
      ...
      M = J2*J4

      ...
    5 K = K+J4
      ...
      MM = J2
      MX = J4
```

Suggested Method

```
      SUBROUTINE PATCH (J1,J2,J3,J4)
      DIMENSION J1(J2),J3(J4)
      ...
      J = J2
      L = J4
      ...
      IF (N.EQ.J) GO TO 5
      ...
      M = J*L
      ...
    5 K = K+L
      ...
      MM = J
      MX = L
```

16.4. MULTIPLY/DIVIDE SEQUENCE

During a calculation of the type N = I*J/K, the intermediate product of I*J may be desired to a length of 72 bits. Subsequent division by K would produce a 36 bit quotient to be stored in N.

When this operation is performed in two statements however, the results may not be desirable. Note the following:

```
M = I*J
N = M/K
```

The product of I*J is truncated to the low order 36 bits and stored in M. The division by K may produce an incorrect quotient if the value of I*J is greater than 36 bits.

The statement is properly written as N = I*J/K, where a 72 bit product is retained for I*J. This is followed by division on the 72 bit dividend by K to produce the proper result to be stored in N.

16.5. REAL TO INTEGER CONVERSION

The following statement is printed in Section 3.2.1 of the FORTRAN IV and V manuals:

> "If a statement implies conversion of an INTEGER constant to a constant which is either REAL, DOUBLE PRECISION, or COMPLEX, or vice versa, then the absolute value of the constant must be less than or equal to $2^{27}-1 = 134,217,727$."

This statement must be observed in all conversions resulting in an INTEGER value. The compiler generated coding does not convert REAL or DOUBLE PRECISION numbers to INTEGER that are within the range of 2^{27} (134,217,728) to $2^{35}-1$ (34,359,738,367).

The assumption is that the rule has been observed, therefore the conversion routine proceeds from a possible "maximum" value of $2^{27}-1$. The effect of this is that a value greater than $2^{27}-1$ would actually be scaled down upon converting it to integer thereby producing an incorrect result.

Where this option is desired, the following subroutine will convert real numbers to integers in the range: 0 to 34,359,738,367.

Call statement:

```
CALL SLIDE (I, REAL, INTGR, $998)
```

```
I = 1, when REAL is single precision
    2, when REAL is double precision
REAL = real or double precision value, positive or
       negative sign
INTGR = signed integer equivalent of REAL
$998 = label of statement for error action when the
       absolute value of REAL exceeds 34,359,738,367.
```

16.5. REAL TO INTEGER CONVERSION (Continued)

Source Coding

```
      SUBROUTINE SLIDE (K,ARG,N,$)
      DIMENSION ARG(2),FP(2),M(2)
      DATA  MX   / 0777777777/
      DATA  HIVAL/ 34359738368./
      DATA  GOVAL/ 134217728./
      DATA    X  / 1.0  /
      DATA   MC  / 0377000000000/
      X = SIGN (X,ARG(1))
      FP(1) = ABS(ARG(1))
      IF(FP(1) - GOVAL) 50,  ,
      IF(FP(1) - HIVAL )  20 , ,
      RETURN 4
   20 M(1) = FP(1).AND.MX
      NC = LSRL((FP(1).AND.MC),27)
      NCD = NC-155
      IF(K.NE.2) GO TO 30
      FP(2) = .ABS(ARG(2))
      M(2) = LSLL(FP(2),9)
      CALL LDSLL (M(1),M(2),NCD,J,NULL)
      GO TO 51
   30 J = LSLL(M(1),NCD)
      GO TO 51
   50 J = FP(1)
   51 IF(X.LT.0) J = -J
      N = J
      RETURN
```

16.6. INTEGER TO DOUBLE PRECISION FUNCTION

In an arithmetic expression, if the mode of one operand is double precision and the mode of the other operand is integer or real, then the mode of the expression is double precision. This means that all operands will be converted to double precision in order to execute the arithmetic expression. When large integer values in the range 134,217,727 to 34,359,738,367, are converted to single or double precision by the FORTRAN IV compiler, there is a loss of significance.

Example:

Convert the integer variable N, containing the value 34,359,738,367, to a real number.

```
X = N
```

The value of X will be 34,359,738,112. The loss of 255 is caused by the limit of 27 bits for the mantissa of a real number.

This loss of significance may not be acceptable in computations involving a series of arithmetic operations with large integer and double precision numbers.

There is no built-in FORTRAN IV function for converting an integer number to double precision where one may retain significance of a large integer number (134,217,727 to 34,359,738,367). The automatic conversion of integer values to the double precision mode by the compiler is carried out as a single precision value with the least significant word set to zero.

The following function may be used in cases where the loss of significance may be a factor in the conversion of large integer numbers to the real mode. This function divides the integer number into two parts - each is less than the capacity of a single precision real number and retains full significance for each part. The two parts are added to produce a double precision sum. This value may then be used with other double precision terms with no loss in significance from the original large integer value.

The name of the function must first be defined as type double precision in the main program:

```
      DOUBLE PRECISION  DP
```

16.6. INTEGER TO DOUBLE PRECISION FUNCTION (Continued)

The coding for the function may be formatted as an internal or external subprogram. For an internal subprogram, N should be typed integer to be made a local variable.

```
      FUNCTION DP(INT)
      INTEGER N
      DATA MSM 10377000000000/
      DATA LSM /   0777777777/
      N = INT
      NSIGN = 1                                    SIGN
      IF(N),5,5                                    SIGN
      NSIGN = 0                                    SIGN
      N = IABS(N)                                  SIGN
    5 I = N.AND.MSM
      DP = I
      J = N.AND.LSM
      DP = DP + J
      IF(NSIGN.EQ.0)DP = -DP                       SIGN
      RETURN
```

The function as written contains fifteen lines. If the sign of the integer value (INT) is not a factor, then the lines labelled SIGN may be removed, leaving ten lines.

Usage Example:

The expression:

```
  XS = N + X
where:  XS = double precision
         N = integer
         X = double precision
```

is now written as:

```
  XS = DP(N) + X
where:  XS = double precision
         N = converted to double precision by the function DP.
         X = double precision
```

16.7. TABLE LOOK-UP TECHNIQUES

16.7.1. DIRECT

The table contains as many entries as the value of the highest numeric table code being processed. The code being searched for literally becomes the index for accessing the corresponding table entry.

Limitations: (1) The codes being processed must contain entirely numeric digits, (2) since the codes usually are not in straight numeric sequence (e.g., 001,002,003,004,etc.) gaps occur between codes and the table size becomes prohibitive and (3) input cards must be read at the beginning of each job to fill the table.

16.7.2. SERIAL

Searching the table beginning with the first table position with each transaction and incrementing the index by one until the condition being tested equals the corresponding table code is a method.

Limitations: (1) When the table size exceeds 20 to 25 entries, the processing time to test all the entries becomes prohibitive, and (2) input cards must be read at the beginning of each job to fill the table.

16.7.3. BINARY

In the binary search approach, the argument to be found is first compared against the middle of the table, and on the basis of this comparison, the upper or lower half will be used for comparison. The table will be halved until an equal condition occurs or the size of the table is reduced to zero. The table arguments must be in sequence. A count of the number of entries (easily generated when the table is filled) is also required.

16.7.4. LITERAL

A table does not appear in the data division, but the codes being searched for appear as literals in the procedure division. A conditional statement must be written for each code to be processed. When the literal being tested equals the transaction code being processed, the corresponding data appears as a literal and is moved to a common storage area.

Limitations: (1) Limited to fixed codes not subject tc change since a literal change requires source language changes and recompilation, and (2) far more coding (programing time) is required than in the other three techniques.

16.7.5. EXAMPLES

16.7.5.1. DIRECT LOOK-UP

1. A variable IV with possible binary values of 0-99 is to be used to pick up a code from a table NTABL of 100 entries using the value of IV as the index for locating the code and placing the code into NV:

```
NV = NTABL (IV +1)
```

2. A variable NX with possible binary values of 0-12 is to be used to recode these values as follows and to store their recode into the variable NY:

```
0-3   recode 1

4-7   recode 2

8-12  recode 3
```

```
    NX = NX+1

    GO TO  (10,10,10,10,20,20,20,20,30,30,30,30,30), NX

10  NY = 1

    GO TO 40

20  NY = 2

    GO TO 40

30  NY = 3

    GO TO 40
```

16.7.5.2. SERIAL LOOK-UP

A variable IV is to be compared with a list NA of 20 values and when a match is made, a tally is to be made in the corresponding position in table NB.

```
    DO   10     J  =  1,20

    IF   (IV - NA(J))      ,20,

10  CONTINUE
    .
    .
    .
20  NB (J) = NB(J)+1
```

16.7.5.3. BINARY LOOK-UP

A variable IV is to be compared with a list NA of 100 values and when a match is made, a tally is to be made in the corresponding position in table NB. The list NA has been sorted using a programer written subroutine called SWAP prior to this procedure. A programer written subroutine called SEARCH delivers a subscript NSUB which may be used as required.

```
      CALL SWAP  (NA,100)
      .
      .
      .
      CALL SEARCH (NA,100,IV,NSUB)          (See 16.16.)

      NB(NSUB) = NB (NSUB) + 1
```

16.7.5.4. LITERAL LOOK-UP

A code NCODE is to be compared against the XS3 literals 0,3,7, and 9 to determine which one it matches. If a match is made, then a recode of 1 is to replace the original code. If no code is found, make the recode 0.

```
      IF  (NCODE . EQ. 3)  GO TO 20

      IF  (NCODE . EQ. 6)  GO TO 20

      IF  (NCODE . EQ.10)  GO TO 20

      IF  (NCODE . EQ.12)  GO TO 20

      NCODE = 0

      GO TO 30

   20     NCODE = 1

   30     CONTINUE
```

16.8. A SUBROUTINE FOR SORTING WORDS IN AN ARRAY

```
      SUBROUTINE SWAP (A,N)
Comment - A is the array of size N
      INTEGER A, TEMP
      DIMENSION A(N)
      N1 = N-1
      DO 20  I = 1,N1
      IF (A(I). LE.A (I+1)) GO TO 20
      DO  10   J = I, 1, -1
      IF  (A(J).LE.A(J+1)) GO TO 20
      TEMP = A (J)
      A(J) = A(J+1)
10    A(J+1) = TEMP
20    CONTINUE
      RETURN
      END
```

16.9. A RANDOM NUMBER GENERATOR ROUTINE (To Generate Numbers as Large as 999)

```
      DATA   T  /O244400000000/
      DATA  MM  /O377777777777/
      K = 2 ** 18 + 3
      IU = LABS (MSR (4HDATE))
      XM = 999
Comment - ABOVE ESTABLISHES INITIAL VALUES
Comment - BELOW GENERATES A RANDOM NUMBER EACH USE
      IU = MM. AND. (IU*K)
      V  = IU/T
Comment - COMPUTE REAL RANDOM NUMBER WITHIN RANGE
      R = XM * V
Comment - FOR INTEGER RANDOM NUMBER IR = XM *V + 1
Comment - XM is upper limit
Comment - IU is start for random number, should be large
Comment - V is real random fraction
Comment - IR is integer random whole number
```

16.10. A SUBROUTINE TO CREATE A CHECK DIGIT FOR A TWELVE-DIGIT NUMBER

```
      SUBROUTINE CHKDIG  (N,M)
      N = 0
      DO  100  I = 12,2,-2
      NT = (IXTRCR (M,I) -3)*2
      IF (NT.GT.9) NT=NT+54
  100 N = JRS (NT,5)+JRS(NT,6)+(IXTRCR(M,I-1)-3)+N
      N = MOD (N.10)+3
      IF (N.NE.3) N=16-N
      RETURN
```

16.11. METHOD USED TO CREATE CHECK DIGIT FOR XS3 FIELD

1. Multiply by 2 the even digits (or odd digits).
2. As each digit is multiplied, if the sum is 10 or more, create 2 digits.
3. Add all digits together and drop the ten's portion of the total.
4. Subtract the remaining unit's position from 10 and the result is the check digit. If the result of step 3 is 0, the check digit is 0.

EXAMPLE: A ten-digit field; create check digit.

8	7	3	5	0	6	9	5	0	5	
	X2 / 14		X2 / 10		X2 / 12		X2 / 10		X2 / 10	
	4	+ 0	+	2	+	0	+	0 =	6	
	1	+ 1	+	1	+	1	+	1 =	5	
	8	+ 3	+	0	+	9	+	0 =	20	

10/31/1 Remainder

10 -1 = 9 the check digit

16.12. A SUBROUTINE TO CONVERT MILLISECONDS TO HOURS, MINUTES AND SECONDS IN ASC *

```
      SUBROUTINE MATIME (MTA)
      INTEGER MTA (3)
      MT = MSR (LSTIME)
      MTA (3) = SIH (IBM MT 3e00000,3),2)+IBM(MOD MT,3e00000)e0000),0)
      MTA (3-1) = SIH(IBM (MOD MOD MT,3e00000),e0000)/1000 ),0),2)
      RETURN
```

16.13. ZERO

```
      SUBROUTINE ZERO (A,N)
      INTEGER A
      DIMENSION A(N)
      DO 10  I=1,N
10    A(I) = 0
      RETURN
      END
```

A subroutine for clearing storage locations.

A: Array name or first location to be cleared.

N: Number of (consecutive) words to be zeroed.

TIMING: 8N + 7e MSEC.

16.14. PERMISSIBLE CALLS USING PROGRAMER'S SUBROUTINE ZERO

16.14.1.

```
      INTEGER X
      DIMENSION X(100)
           ...
      CALL ZERO (X,100)          Clears all of array X.
           ...
      CALL ZERO (X,50)           Clears first 50 locations.
           ...
      CALL ZERO (X(1),50)        Clears first 50 locations.
           ...
      CALL ZERO (X(25),30)       Clears X(25) through X(54).
           ...
      CALL ZERO (X,150)          Clears all of X plus 50
                                 locations following X.
```

*A subroutine CLOCK is available which will typeout in FIELDATA "REAL TIME CLOCK INTERROGATED AT xxxxxx."

16.14.2.
```
INTEGER BB
DIMENSION  BB(5,6)
     ...
CALL ZERO (BB,30)                    CLEARS ALL OF BB.
     ...
CALL ZERO (BB,5*6)                   CLEARS ALL OF BB.
```

16.14.3.
```
PARAMETER N1 = 100,N2 = 10,N3 = 25
DIMENSION MAD(N1,N2,N3)
     ...
CALL ZERO (MAD,N1*N2*N3)             In each case, entire
                                     array will be cleared.
     ...
CALL ZERO (MAD,100*10*25)
     ...
CALL ZERO (MAD,25000)
```

16.14.4.
```
INTEGER VAR                          Note:  VAR is a
                                            variable here --
     ...                                    NOT an array.
CALL ZERO (VAR,1)                    CLEARS VAR.
     ...
CALL ZERO (VAR,10)                   CLEARS VAR AND NEXT
                                     NINE LOCATIONS.
```

NOTE:

1. Subroutine expects an array name, but starting location may be given in calling statement.
2. A variable may be given instead of an array in CALL statement.
3. Subroutine is set up in terms of a one-dimensional array, but more-than-one-dimensional arrays may be cleared with the routine.

CAUTION:

1. If you are clearing less than an entire array which is more than one-dimensional, be sure you are clearing the area you intend.
2. If value of N exceeds dimension limits, be sure what the following locations are.

16.15. SEQUENCE CHECKING

Many problems require that the input data be arranged in a logical sequence (ascending or descending) according to designated key words within an item. For example, a file of data may be required to be in state, county sequence. Before processing such data, it is often required as a safeguard that a check be made to insure the fact that a good sorted file is being processed. Therefore, the need to sequence check. The Arithmetic IF is made to order for this kind of checking. An example of sequence checking appears below. Note that if there is a break in sequence, an error condition is noted and the program terminates by calling a programer written subroutine called DUMP containing the single statement RETURN 0.

```
      IF (MAJORP - MAJORN)  30,    , 20

      IF (MINORP - MINORN)  30, 30, 20

   20 CALL TYPE (20, 20H SEQUENCE CHECK FAILS)

      CALL DUMP

   30 CONTINUE

      .

      .

      .

      SUBROUTINE DUMP

      RETURN 0
```

16.16. SEARCH: PROGRAMER'S SUBROUTINE FOR BINARY SEARCH

Purpose -- The purpose of this routine is to search a sequenced list of single-word items for equality against a given item or argument. If a match is found, the subscript location where equality occurred in the list is placed in a flag cell. If no match is found, the flag cell will be zero. There will be an execution time advantage over the serial scan method if the list size is 35 or greater.

Method -- The argument to be found is first compared to the content of the midpoint in the list of items. On the basis of this comparison, the upper or lower half of the table is used to continue the search. The list will be successively halved until an equal condition occurs, or the size of the list reduces to zero.

16.16. SEARCH: PROGRAMER'S SUBROUTINE FOR BINARY SEARCH (Continued)

Coding

```
      SUBROUTINE SEARCH (LIST,N,NFLD,NSUB)
      INTEGER HE,LE,S
      DIMENSION LIST(N)
      HE = N+1
      LE = 0
10    S = LSRL  ((HE+LE),1)
      IF (NFLD-LIST(S))  15,12,11
11    LE = S
      GO TO 20
12    NSUB = S
      RETURN
15    HE = S
20    IF  ((LE+1)-HE)  10,  ,10
      NSUB = 0
      RETURN
```

Definitions

LIST = Sequenced array of single-word items to be searched.

N = Number of single-word items in LIST.

NFLD = Argument searched for in LIST.

NSUB = Flag cell containing the subscript location where equality occurred in LIST, or zero when argument cannot be found in list.

16.17. DRUM USE

16.17.1. FH*80DRUM

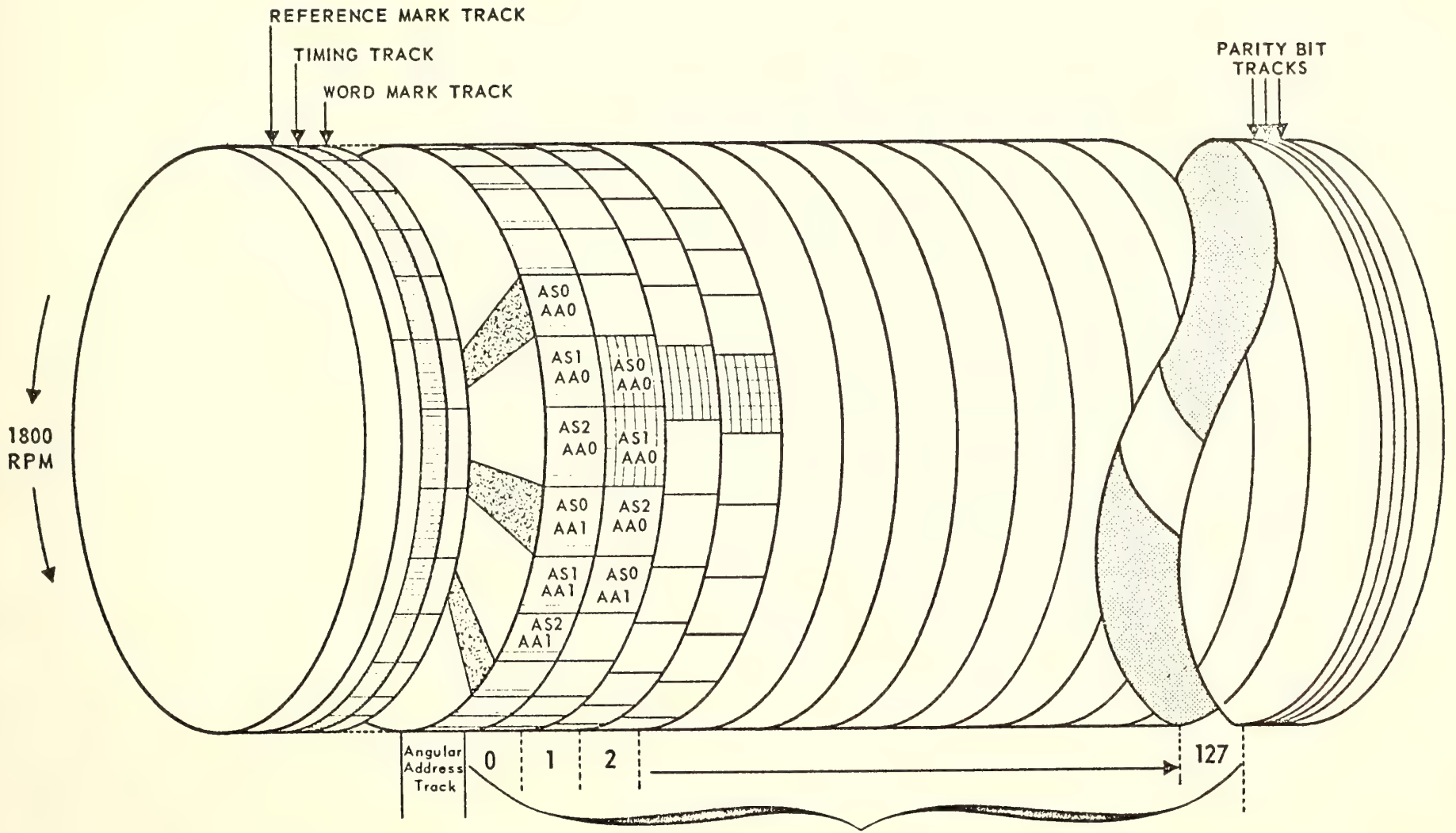
16.17.2. FOLION

16.17.3. EX

16.17.4. SCHEDULE CARDS

16.17.5. PRACTICAL USES OF THE DRUM

16.17.6. SEGFOR - DRUM OVERLAY SEGMENTATION



BAND = 3 ANGULAR SECTIONS
6 TRACKS
6144 ANGULAR ADDRESSES

ANGULAR SECTION = 2048 ANGULAR SEGMENTS.

16.17.1. FH880 DRUM

The drum is a storage device; it is not part of the central computer. The Census FH880 configuration has two drum units each with a capacity of 786,432 computer words (a total of 1,572,864). After the initial access time (average 17 milliseconds) required to find the first location requested in the drum, the transfer rate is 16.5 microseconds per word. Therefore, the time required for a sizable transfer is not much more than for one word.

The Census FH880 drum subsystem consists of a channel synchronizer, a control unit, a power supply and two drum units. The drum and head subassemblies are mounted in the drum cabinet. The drum is an oxide coated cylinder, 30 inches long and 24 inches in diameter.

Forty head blocks are positioned around the drum. Mounted in each head block are 22 read/write heads, or 880 heads in all. Each head is associated with one track around the drum. There are 128 6-track recording bands across the drum, accounting for 768 data tracks. Some of the remaining tracks are used for parity, reference, and timing functions. The rest are categorized as spares. Each band is further divided into 2048 angular addresses, around the circumference of the drum. Each address is further divided into three angular sections. The angular section, within any single angular address, represents the location of a 36-bit word. This scheme provides 6144 logical addresses for each band on each drum unit. The 36-bit data word is stored in the form of a 6 x 6 array. The array results from a combination of 6 rows and 6 tracks of data. Each of the three sections of a drum, instead of constituting a single entity, is divided into 2048 segments, one for each angular address, and these segments are arranged around the periphery of the drum in a repetitive pattern: AS0, AS1, AS2..etc. Addresses for drum unit 0 range from 0 to 02777777. Addresses for drum unit 1 range from 03000000 to 05777777.

The three designators for forming the address of the drum are as follows:

	Decimal	Octal
Angular Section (Drum 0- 00-02) (Drum 1- 03-05)	00-05	00-05
Band Number	000-127	000-177
Angular Address	0000-2047	0000-3777

The format required by the computer is shown in the following chart:

	Angular Section	Band Number	Angular Address
Bits	22-18	17-11	10-0
Octal	0 5	1 7 7	3 7 7 7
Bits	00 101	1 111 111	11 111 111 111
Address	0 5	7 7 7	7 7 7

The FH880 Magnetic Drum Control Unit controls the flow of information between the drum units and the Channel Synchronizer. It will accept 6-bit output data characters from the synchronizer and distribute them to the drum units as specified by the program. The Control Unit also transmits 6-bit input data characters to the Channel Synchronizer. Transfers between drums and core memory begin at the time the angular position of the drum coincides with the specified starting address.

Several errors are detected by the Magnetic Drum Subsystem:

Parity Error. Each data word is recorded with odd parity. If even parity is detected during the read operation, a parity error Interrupt occurs.

Character Count. Whenever fewer than six characters per word are transferred to or from the drum, a character count error interrupt takes place.

Illegal Function. Each operation requested of the drum subsystem is checked for validity.

Invalid Address. Any address beyond the capacity of the drum subsystem results in an invalid address interrupt.

16.17.2. FOLION

Drum files may be read or written sequentially in the same manner as tape files. However, a maximum of 700,000 cells are allowed per program. When a file is closed with a rewind, it may be reopened and read or written from the beginning of the file area.

3. EX

Drum files may be read or written in either a sequential or random manner. The programmer must specify exactly how many words to read or write and where they are located. The programmer has complete control and the responsibilities for blocking, item control, etc. The EX subroutine merely delivers n number of words from a specified relative location into a specified location.

The CALL line of the EX routine is:

CALL EX(function,unit,status,buffer,no.words,relative drum position

The most commonly used drum functions are:

35 - read drum

02 - write drum

The unit assignments for drum are:

25 for drum table D1 with length tag LD1

26 for drum table D2 with length tag LD2

27 for drum table D3 with length tag LD3

28 for drum table D4 with length tag LD4

29 for drum table D5 with length tag LD5

The status variable will contain the status code of the completed function. It should be checked after every CALL to the EX subroutine. If the code is Ø, there was normal completion.

The buffer is the area in core into which data will be read from drum or from which data will be written into drum.

The number of words is the number of words to be transferred in either direction. A block size of 256 words is near optimum in timing for drum transfers.

The drum position (relative to position Ø) is the position within the drum table where reading or writing will take place. For example, if the drum position is Ø, then the first position in the drum table is the starting position of the transfer. If the drum position is 99, then the starting position is the 100th word. The programmer must maintain the variable which specifies the drum position. The EX subroutine does not update the variable.

To read a series of 20 word groups from drum (unit 25) and write them out using FOLION, the following could be coded:

```
      DIMENSION NBUFF(2Ø), NA(6Ø5)
..... coding to initialize FOLION tape .....etc.
      NN=Ø

      DO 5Ø J=1,1ØØ

      CALL EX (34,25,NSTAT,NBUFF,2Ø,NN)

      IF (NSTAT.NE.Ø) STOP ERROR

      CALL ITMOUT (6HANYFIL,NBUFF)

   5Ø NN=NN + 2Ø
```

16.17.4. SCHEDULE CARDS

If no drum is used, the FAC card should contain MDØ/Ø. If drum is used, the total number of drum cells required should be included as MDØ/xxxxxx. Also, a TAL card should be furnished using the drum length tags and specifying the size of each area. For example.

,TAL, LD1/5ØØØØ, LD2/1ØØØØØ

16.17.5. PRACTICAL USES OF THE DRUM

The drum may be used as a scratch pad: to store intermediate files (to conserve tapes and time), to store tally matrices, to disperse files (to eliminate the need for sorting short files), to have random access to information based upon indexed keys in core, or to help in any other way the programmer can envision. In some instances, master files are read in and stored on the drum for unlimited access. Program segmentation is also feasible on the drum.

Random Access, Indexed Secuential

One example of drum use is:

It is desired to store information in the drum and to access records in a random manner. At the time the record is stored, a key and a relative location of drum storage can be placed in a core table. This core table can be sorted, scanned, or processed sequentially or randomly. When the desired key is selected in core memory, then the relative drum location can be placed in a variable before a call to EX. Information may be stored into or read from selected areas in drum. This type of operation should be performed with discretion because of time factors. Too many references to small areas in drum can slow down a program. It is best to use chunks of data in the range of 256 words per reference.

Tally Matrices

One use of the drum is to store the matrices developed during a program run. After the data for an area has been summarized at the detail level, the matrix for that specific area can be stored. When all matrices have been stored, it is a simple matter to code a loop to read each matrix, to create cross totals, subtotals and grand totals for all like matrices, and then to produce tables. The same matrices may be used to prepare many different tables depending upon specifications.

Dispersal by Selection

The drum is ideal for dispersing a file into several smaller ones based upon special criteria either before or after initial processing. It often eliminates the necessity for a sort especially if the data is a short file of less than 700,000 words. When there is room in core and when there is a need for a sort, a selected drum area (which has been dispersed) can be brought into core and sorted using the subroutine SORT. The sorted data may then be sent back to drum and later merged with other short drum files.

Parameters for Program Execution

A program technique is to load the drum with a long list of parameters or selected data which will be tested in the program to direct the sequence of operations, to furnish table specifications, to furnish keys for operation, etc.

Master File Storage

When a master tape file of whatever nature (heading, stubs, alphabetical description, codes, table formats, etc.) is needed repeatedly throughout a run, it is definitely an advantage to be able to store this file on drum and to read it either sequentially or randomly as the need arises. The EX routine can be used for this purpose.

Program Segmentation

It is possible to assemble several independent programs and to call each one as needed into an overlay area for execution under the control of a main segment. See SEGFOR 16.17.6.

16.17.6. SEGFOR

DRUM OVERLAY SEGMENTATION OF FORTRAN CODED PROGRAMS

Bureau of the Census - Paul Friday

DRUM OVERLAY SEGMENTATION:

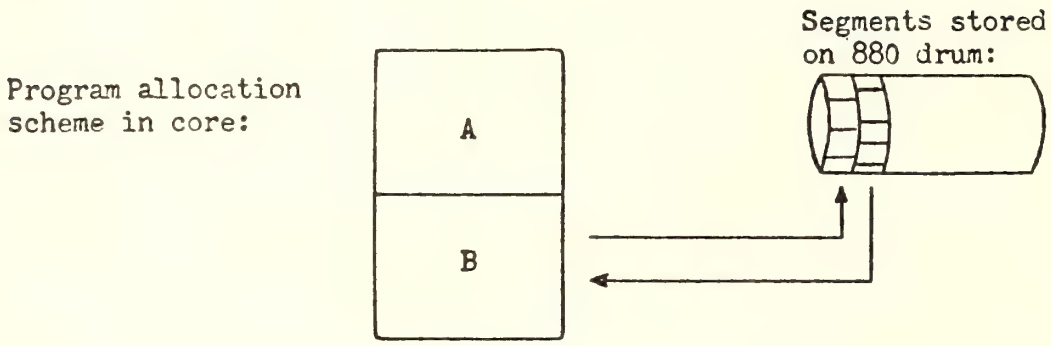
- A program may be written in self contained pieces or "segments".

- When this program is loaded into the machine, each of these segments reside on drum rather than in core, and may be read into core one by one for execution under control of the running program itself.

- Each segment, when called for, will be read into a common core area for execution. This allows any number of segments (up to a maximum of 9) while the core requirements for the overlay area are defined only by the requirements of the largest segment.

- The segments are initially coded as free standing independent programs and become part of a master segmentation program when they are loaded into the machine. For this reason, each segment may be tested as an independent program and then, with slight modification, be included as part of the segmented program as a final step. Also, former sequences of independent programs may be modified and defined as drum segments of a single program. This is useful in eliminating intermediate tape handling and operational errors.

Pictoral layout of a segmented program:



- The above diagram applies to both IBANK and DBANK.

- Section "A" consists of the resident segment which is in core at all times. Section "A" contains your resident Fortran coding plus any library routines referenced in that section.

- Section "B" initially contains nothing. Section "B" is the area into which the various drum segments will be read and then executed. Each segment read into "B" consists of your Fortran coding for that segment plus any library routines referenced from that segment that were not previously referenced in the resident section (see below).

Global and Local Library Routines:

- If any drum segment references a library routine (such as Folion) that was also referenced in the resident coding, the reference will be to the library routine allocated in the resident core slot. Thus, any library routine referenced from the resident section will be known as a global routine.

- If any drum segment references a library routine that was not referenced in the resident coding, the routine will be allocated in each and every segment that makes reference to it. Thus, any library routine not referenced in the resident section but referenced in a drum segment (or segments) will be known as a local routine.

Coding of the segments:

Each segment in the program, including the resident segment, is coded in the following format:

```
/././.*     NAME
@ MAP NAME,NAMEB.
@LIP FOR NAME,NAME,NAMEB.
      CALL SUB
      SUBROUTINE SUB ($,$,$,$,$,$,$,$,$)
           *
           *
      (YOUR FORTRAN STATEMENTS ARE CODED HERE)
           *
           *
      RETURN n        (SEE FOLLOWING DISCUSSION OF RETURNS)
      END
 ))))))
```

Note that additional segments are coded as separate files on the same tape, each of which is equivalent to an independent program. Only the start card of the last file should contain the REWINT entry in card columns 73-78.

- When the segmented program is initiated, the resident segment is given control first.

- Reading drum segments into core is controlled by the execution of RETURN statements. The legal forms are RETURN and RETURN n (where n is coded as an integer ranging from 1 to 9).

- The RETURN n statement will cause the drum segment described by n to be read into core (if it is not already there) and executed. The segment previously residing in core is written back to its proper slot on the drum. For example, execution of a RETURN 3 statement from any segment in the program will read segment 3 into core (if it is not already there) and execute it.

- The RETURN statement executed from any segment in the program will cause control to be transferred to the resident segment.

- Each segment read in from drum is charged to the program as an I/O request. In other words, shuttling between the segments is not free. Each segment read from drum charges 8 u sec per word in the segment to the running time of the program. Therefore, the program should be designed to avoid excessive shuttling between the drum segments. However, if the segment to be read from drum is already in core, no I/O action is required. For example, shuttling between the resident segment and any one drum segment over and over would require no drum action.

COMMON AREAS

- Any or all of the segments may contain blank COMMON statements.

- The area generated by these statements is truly common - that is, data placed in blank COMMON by any segment is available for use in all other segments containing similar blank COMMON statements.

- However, no matter where the blank COMMON statements occur, the COMMON area is allocated in the resident segment core area and never becomes part of any drum segment.

- Only blank COMMON (never named COMMON) may be used in a segmented program.

A Further Discussion of Global and Local Library Routines

- The concept of global and local library routines must be considered when setting up a segmented program.

- A global library routine is defined as one that is referenced from the resident segment. If that same library routine is referenced from a drum segment, the reference will be to the routine allocated in the resident core area. For example, if a reference is made to Folion from the resident segment (defining it as a global library routine), then any files opened in any segment may be referenced from any other segment in the program.

- A local library routine is defined as one that is referenced from a drum segment or segments but not from the resident segment. In this case, a separate copy of the library routine will be allocated in each drum segment that makes reference to it. For example, if Folion is defined as a local routine, files opened in any one drum segment may not be referenced from any other segment in the program. However, the other segments may maintain unique files on separate units.

- In order to allocate a library routine in the resident segment (defining it as a global routine), it is only necessary to make a reference or call to it. It is not mandatory that referencing statement ever be executed.

- The following library routines, because of their design, may be allocated in only one spot in the segmented program. This means that they may be either locally defined in only one drum segment, or that they may be globally defined. The routines are:

    1) the double precision library package
    2) Fortran formatted writes on units -2,-3, and +6

The double precision routine is referenced by writing a DOUBLE PRECISION type statement. The Fortran formatted write routine is referenced by formatted write statements involving one or more of the units described above.

SCHEDULING

PTY CARD: The program name is SEGFOR. (The Fortran coded segments become load time subroutines to the SEGFOR controlling program.)

FAC CARD: The IBANK requirement is:

The resident segment IBANK + the largest drum segment IBANK + 250_{10} (or 327_8). Note that the IBANK requirements for each drum segment are not the same as shown on the Elf listing. Any library routines located in the resident segment will be global and therefore not allocated in drum segments which reference them. Because of this, the core requirements for each global routine must be subtracted from each drum segment which references that global routine. For example, if Folion were a globally defined routine, and a drum segment contains Folion on its Elf listing, then the core requirements of Folion must be subtracted from the drum segment requirements shown on the Elf listing.

- The DBANK requirement is:

  The resident segment DBANK + LN$BLN + the largest drum segment DBANK + 250_{10} (or 372_8). LN$BLN is a TAL card increment explained later. Note that the comments on drum segment IBANK apply to the DBANK computations as well.

- The Drum requirements are:

  LD1 + LD2 + LD3 + LD4 + LD5 + sufficient drum to store IBANK and DBANK of all drum segments. LD1 through LD5 are TAL card increments explained later.

- Tape facilities are scheduled in the standard manner for Fortran coded programs.

TAL CARD:

LN$BLN — This entry is equal to the value which appears on the Elf facility synopsis opposite "LN$BLN". The entry on the TAL card should be equal to the largest LN$BLN on the Elf listings for the segmented program.

LD1 through LD5 — These represent the standard Fortran drum table lengths, if used. They correspond to Fortran internal unit numbers 25 through 29.

SUB CARD

SØ is the formal name of the resident segment.

S1 through S9 are the formal names of the first through the ninth drum segments respectively.

To fill out a SUB card, you must equate the formal names of the segments with the second name appearing on the MAP card of each segment. For example, a program containing a resident and two drum segments would be scheduled with a SUB card as follows:

JRID,SUB,SØ/NAM1,S1/NAM2,S2/NAM3,S4,S5,S6,S7,S8,S9.

Where NAM1, NAM2, and NAM3 are the second names on the MAP card of each segment.

Note that the SUB card is the only point in the system where the elements are formally defined as drum segments. Because of this, it is possible to construct a tape containing any number of elements which could be used as segments. Then, when the program is scheduled, the SUB card may describe various configurations of the segmented program.

For example, your program tape could contain the following named elements (2nd name on the MAP card):

NA,NB,NC,ND,NE,NF, and NG

Then, one possible configuration of the program defined at schedule time might be:

JRID,SUB,SØ/NA,S1/NB,S2/NF,S3/ND,S4,S5,S6,S7,S8,S9.

And then, without recompilation, the program configuration could be redefined simply by rescheduling with a different SUB card:

JRID,SUB,SØ/NA,S1/NG,S2/NC,S3,S4,S5,S6,S7,S8,S9.

TRIVIA

- The coded example shown on page 3 may contain references to other internal subroutines or other external subroutines contained within the SERVO start and stop cards.

- The "CALL SUB" statement is never actually executed when the element becomes part of the segmented program. However, it must be present for loading purposes. The first statement within subroutine "SUB" will be the initial statement executed in the segment.

- Note that the "CALL SUB" line could contain return statement number arguments referring to coding following the "CALL SUB" statement. Although this coding would never be executed when the element is part of a segmented program, it would be executed if the element is tested as an individual program. By coding it this way, the same element may be tested either as an independent program or as a segment of another program without recompilation modifications.

- Segmentation is valuable not only because it offers the ability to code larger programs, but also because it allows the construction of a system which can be varied by SUB card redefinitions at run time.

- Segments may be added to an existing segmented program tape at any time without touching the rest of the program. This allows the construction of an open-ended system similar in concept to the SORT with its ability to accept option coding subprograms.

- Any requests to Special Services to compile a segmented program must indicate that a "FORELF-ELF" compiling sequence is desired.

- An initial typeout indicating the FORTRAN program's name (the main program name is always SEGFOR) and version number is recommended since control is more difficult with programs of this type.

- For practical purposes the number of program elements on a tape should be less than 100. It is possible to design segments in such a way that several may be considered main segments. Then, at load time one of the main segments may be sleected along with the other segments desired for a run. Thus, similar but different programs may be executed depending upon the need.

- The following information is contained in a word within SEGFOR:

    H1 - number of cells in the overlay segment
    H2 - address of the current overlay segment less one

  The location of this word may be found in a memory dump by adding the last address of the ELF allocation of the main segment, the loading address of the IBANK, and 133 octal.

- The program elements reside in core storage as follows:

    Main segment
    SEGFOR
    Overlay area

17. PROGRAM EXAMPLES

17.1. GENERAL

The sample programs which follow have been coded from training problem specifications. The reader should note that there are many ways to code for the solution of a problem. The examples shown here are merely used as illustrations.

The control cards required for the assembly of a program are not illustrated here. Refer to Chapter 12: "ASSEMBLY, CORRECTING, AND DEBUGGING.

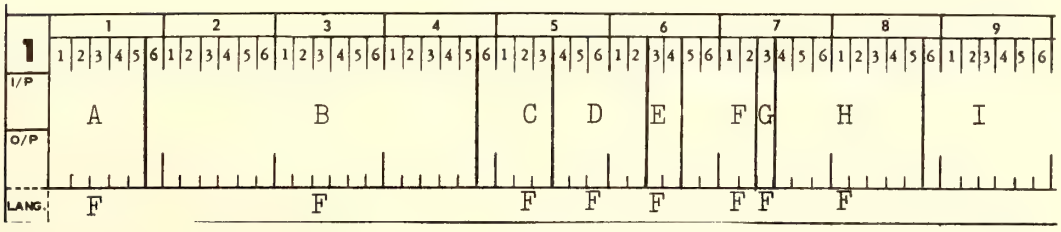
17.2. PAYROLL PROBLEM SPECIFICATIONS

The input reel for this problem is a IIIA in LION External Format (Fieldata card images). The last record contains all 9's.

The output reel should contain a table showing for each employee the following: employee number, name, hours worked this week, number of overtime hours worked, gross pay, amount of overtime pay, amount of withholding tax, retirement deductions, amount deducted for health insurance premium, net pay, and pay period. Appropriate headings should be on each output column. The payroll will be computed for a week's work. The regular work week will be 40 hours. Any hours worked above this will be considered overtime hours. Retirement is 3% of gross for all employees.

The data fields are described as follows:

FIELD	DESCRIPTION
A	Employee number
B	Employee name
C	Regular hourly rate, 2 implied decimal places
D	Overtime hourly rate, 2 implied decimal places
E	Number of dependents, integer field. The withholding tax is computed on the following basis: 0 to 1 dependents, tax is 20% of gross 2 dependents, tax is 15% of gross 3 dependents, tax is 10% of gross 4 dependents, tax is 5% of gross 5 or more dependents, no tax is withheld
F	Total number of hours worked, 1 implied decimal place
G	Health Insurance Option, integer: 0 means no insurance 1 means low option, $1.75 per pay period 2 means high option, $2.85 per pay period
H	Closing date of the pay period in form MO/DA/YR
I	Field I is not used.



17.2. PAYROLL PROBLEM SPECIFICATIONS (Continued)

Payroll Coding

```
   DIMENSION TAX(6),H(2),B(3)
   DATE TAX /.2,.2,.15,.1,.05,.0/
10 FORMAT(A5,3A6,F4.2,F5.2,I2,F4.1,I1,A6,A2)
20 FORMAT (1H /,
  A40H NUMBER       NAME         NO. HOURS  OV,
  B40H.TIME GROSS PAY  OVER. PAY   TAXES   RET,
  C40HIREMENT    INSUR.   NET PAY   PAY PERIOD//)
30 FORMAT (1X,A6,3A6,4X,F4.1,6X,F4.1,4X,3(F5.2,6X),
  A F5.2,7X,F4.2,6X,F5.2,5X,A6,A2/)
 1 READ (5,10) A,B,C,D,I,F,J,H
   IF (I.EQ.99) GO TO 999
   WRITE (6,20)
   Q=F-40.0
   IF (Q.LT.0.0) GO TO 35
   Z=40.0
   GO TO 36
35 Q=0.0
   Z=F
36 Q=D*Q
   P=C*Z+Q


   I=I+1
   IF (I.GT.5) I=6
   R=TAX(I)*P
   S=P*.03
   IF (J-1)  ,50,60
   T=0.0
   GO TO 70
50 T=1.75
   GO TO 70
60 T=2.85
70 U=P-R-S-T
   WRITE (6,30) A,B,F,D,P,Q,Q,R,S,T,U,H
   GO TO 1
999 STOP
   END
```

17.3. PROBABILITY OF PURCHASING AN AUTOMOBILE

Prepare the following table:

PROBABILITY OF PURCHASING AN AUTOMOBILE WITHIN ONE YEAR BY CONDENSED INCOME BY AGE OF HEAD OF HOUSEHOLD

	PROBABILITY OF PURCHASE											
Income	Total	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1.0	NA
Total													
Under $3000													
$3000-4999													
$5000-7499													
$7500-9999													
$10,000 & Over													
NA													

Number should be to nearest thousands

Repeat for Age of Head: Under 25
25-34
35-44
45-54
55-64
65 +
NA

Skip non-interview records; i.e., limit universe to records with a code Ø in Word 39, Character 1.

17.3. PROBABILITY OF PURCHASING AN AUTOMOBILE (Continued)

Necessary information can be found as follows:

	DESCRIPTION
Word 9 Characters 1-2	AGE of Head of Household (Single years from 15; NN for NA)
Word 9 Characters 5-6	INCOME

Code

01	UNDER $1000
02	$1000-1999
03	2000-2999
04	3000-3999
05	4000-4999
06	5000-5999
07	6000-7499
08	7500-9999
09	10000-14999
10	15000-24999
11	25000 and over
NN	NA

Word 28 Character 3

PROBABILITY OF PURCHASING AN AUTOMOBILE WITHIN 1 YEAR

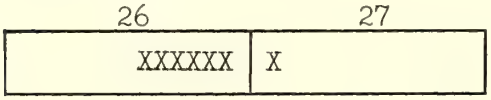
Code

Ø	Ø
1	.1
2	.2
3	.3
4	.4
5	.5
6	.6
7	.7
8	.8
9	.9
A	1.0
N	NNA

Word 26 Characters 1-6

Word 27 Character 1

WEIGHT (Each person is relatively similar to the society in general; these weights, rather than 1 are added within the table. There are two assumed decimal places.)



17.3 PROBABILITY OF PURCHASING AN AUTOMOBILE (Continued)

FORTRAN CODING -

```
      DIMENSION NWS(640),NWDS(640),NTABL(20,11),
     A ITEM(60),N25(7),NHDR(20),NAGE(2,10)
      INTEGER TABL(13,7,8)
      DATA NA,N25/05050,0,25,35,45,55,65,200/
      DATA((NTABL(I,J),I=1,20),J=1,9) '                                  60H
     1                                          PROBABILITY OF  ,60H
     2 PURCHASE                                                  ,60H
     3                                                           ,60H
     4                                                           ,60H
     5INCOME            TOTAL   0    .1    .2    .3    .4    .5  ,60H
     6  .6    .7    .8    .9    1.0   NA                          ,60H
     7                                                           ,60H
     8                                                           ,60H
     9TOTAL                                                      ,60H
     1                                                           ,60H
     2UNDER $3000                                                ,60H
     3                                                           ,60H
     4$3000-4999                                                 ,60H
     5                                                           ,60H
     6$5000-7499                                                 ,60H
     7                                                           ,60H
     8$7500-9999                                                 ,60H
     9                                                            /
      DATA((NTABL(I,J),I=1,20),J=10,11)                          /60H
     1$10,000 & OVER                                             ,60H
     2                                                           ,60H
     3NA                                                         ,60H
     4                                                            /
      DATA NAGE                                                  /60H
     1 TOTAL       UNDER 25    25-34       35-44       45-54     ,60H
     2 55-64       65+         NA                                 /
      DATA NHDR                                                  /60H
     1 PROBABILITY OF PURCHASING AN AUTOMOBILE WITHIN ONE YEAR BY  ,60H
     2AGE OF HEAD OF HOUSEHOLD                                    /
C
C PROGRAM PROCEDURES START HERE
      CALL OPNIN (3HNAS, NWS, 640, 10, 0, 0, 0, 0, 0)
      CALL OPNOUT (3HOUT, NWDS, 640, 20, 11, 0, 0, 0, 0)
      DO 1 I=1,20
      DO 1 J=1,11
```

17.3. PROBABILITY OF PURCHASING AN AUTOMOBILE (Continued)

FORTRAN CODING (Continued)

```
    1 NTABL(I,J)=LFX(NTABL(I,J)
      DO 10 I=1,20
      NHDR(I)=LFX(NHDR(I))
   10 NAGE(I,1)=LFX(NAGE(I,1))
      CALL INSERT (31,NHDR,1)
      CALL ITMOUT (3HOUT,NHDR)
      CALL ITMOUT (3HOUT,NTABL(1,2))
      CALL ITMOUT (3HOUT,NTABL(1,2))
  11  CALL ITMIN(3HNAS,ITEM,N,$999)
C  CHECK TO SEE IF THERE WAS AN INTERVIEW
      IF (JRS(ITEM(39),1).NE.3) GO TO 11
C
C  RECODE AGE
      IF (JRT(ITEM(9),1).EQ.NA) GO TO 2Ø
      KAGE=LXB(JRT(ITEM(9),1))
      DO 15 K=2,7
      IF (KAGE-N25(K))21, ,
  15  CONTINUE
      GO TO 21
  20  K=8
C
C  RECODE INCOMES
  21  IF (JNT(ITEM(9),3).EQ.NA) GO TO 3Ø
      INC=LXB(JNT(ITEM(9),3))
      GO TO (22,22,22,23,23,24,24,25,26,26,26),INC
  22  J=2
      GO TO 31
  23  J=3
      GO TO 31
  24  J=4
      GO TO 31
  25  J=5
      GO TO 31
  26  J=6
      GO TO 31
  3Ø  J=7
C
C  RECODE PROBABILITY
  31  NPROB=JRS(ITEM(28),3)
      IF (NPROB.EQ.2Ø) GO TO 32
      IF (NPROB.EQ.4Ø) GO TO 33
      I=NPROB-1
      GO TO 4Ø
  32  I=12
      GO TO 4Ø
  33  I=13
```

17.3. PROBABILITY OF PURCHASING AN AUTOMOBILE (Continued)

```
C  GET WEIGHTS
  40  NCUM=LXB(ITEM(26))
      NCUM=NCUM*10+LXB(JRS(ITEM(27),1))
      TABL(I,J,K)=TABL(I,J,K)+NCUM
      GO TO 11
C
C  AT END OF FILE,COMPUTE TOTALS
 999  DO 51 K=2,8
      DO 50 J=2,7
      DO 50 I=2,13
  50  TABL(1,J,K)=TABL(I,1,K)+TABL(I,J,K)
      DO 51 J=2,7
      DO 51 I=1,13
  51  TABL(I,1,K)=TABL(I,1,K)+TABL(I,J,K)
      DO 60 K=2,8
      DO 60 I=1,91
  60     TABL(I,1,1)=TABL(I,1,1)+TABL(I,1,K)
      DO 70 I=1,728
  70  TABL(I,1,1)=LBX((TABL(I,1,1)+50000)/100000 ,0)
C ABOVE ROUNDS AND DIVIDES BY 1000.00
C  PREPARE THE OUTPUT TABLE
      DO 100 K=1,8
      NTABL(1,2)=NAGE(1,K)
      NTABL(2,2)=NAGE(2,K)
      DO 62 NK=1,4
  62  CALL ITMOUT(3HOUT,NTABL(1,NK))
C
      DO 64 J=1,7
      DO 63 I=1,13
  63     NTABL(I+3,J+4)=TABL(I,J,K)
  64  CALL ITMOUT(3HOUT,NTABL(1,J+4))
 100  CONTINUE
      CALL IENFIL(3HNAS)
      CALL OENFIL(3HOUT)
      END
```

17.4. SORT WITH OPTION CODING

FILE DESCRIPTION:

Input: Size - variable length records; largest is 80 words
Language - all in BCD-XS3
IIIa tape, written with FOLION and buffer of 640 (605-LION) words
Format - see attached record description

Output: Sort output is to be blanked
IIa output 1 -- print on off-line printer, board 3, loop 3, paper 143-1
IIa output 2 -- (same as above)

GENERAL DESCRIPTION:

Type out: TRAINING SORT PROGRAM

List: B-C Type Units; bypass all others. (Type B-C is determined by REC TYPE = 3, and SAMPLE = 1

Sort by: PSU, SERIAL, and SEGMENT as follows:

Output 1: Vacant, Year-round Housing Units

Year-round = 1 in "Seasonal Status"

Housing Units = 1 through 7 in "Type Qtrs"

Create internal label - Vacant Units Sorted : external label - Vacant

Format headings:

PSU SERIAL SEGMENT TYPE-QTRS VAC. STAT SEAS. STAT

Output 2: All other units (not vacant and/or not year-around and/or not housing units)

Create internal label - Not Vacant Units Sorted : external label - Not Vac

Format headings:

PSU SERIAL SEGMENT TYPE-QTRS VAC STAT SEAS STAT USUAL OCC COND

17.4. SORT WITH OPTION CODING (Continued)

INPUT RECORD OF FIRST PASS

WORD	DIGIT(S)	DESCRIPTION	COMMENTS
1	1	Record Type	Accept "3", reject all others
1	3	Sample	Accept "1", reject all others
1	4,5,6	PSU	Major sort key
2	1,2,3,4	Serial	Intermediate sort key
5	1,2,3	Segment	Minor sort key
7	1,2	Vacancy Status	If vacant, codes "01,02,03"
8	2,3	Type QTRS.	If housing unit "01-07"
48	3	Seas. Status	If year round, code "1"
48	4	Usual occ.	Display code if NOT VAC file
48	5	COND.	Display code if NOT VAC file
49-80	Other Information		

OUTPUT RECORD OF FIRST PASS

All fields are right aligned with ignore fill to the left.

WORD		
1	Record Type	
2	Sample	
3	PSU	
4	Serial	
5	Segment	
6	Vacancy Status	Accept 01-03 as vacant
7	Type Quarters	Accept 01-07 as Housing Unit
8	Seas. Status	Accept 1 as year round
9	Usual occ.	Display code if NOT VAC
10	Condition	Display code if NOT VAC

17.4. SORT WITH OPTION CODING (Continued)

FIRST PASS CODING FOR SORT PROBLEM

```
   EXTERNAL FPS
 1 CALL OCSUB ($1,$1,FPS)
   SUBROUTINE FPS ($,$,$)
   COMMON NENTRY,ITMSIZ,ITEM(80),NEA(14),NTEM(10)
   DATA NEA/1,-13,1,3,4,-51,3,-71,2,-82,2,-483,31,0/
   IF (NENTRY) 20,20,30
20 RETURN 1
30 IF (K.EQ.0) CALL TYPE (21,21HTRAINING SORT PROGRAM)
   K=1
   IF (JRS(ITEM(1),1).NE.6) RETURN 1
   IF (JRS(ITEM(1),3).NE.4) RETURN 1
   CALL UNPK (ITEM,NEA,NTEM)
   DO 31 J=1,10
31 ITEM(J)=NTEM(J)
   ITMSIZ=10
   RETURN 3
   END
```

17.4. SORT WITH OPTION CODING (Continued)

LAST PASS CODING FOR SORT PROBLEM

```
   EXTERNAL LPS
 1 CALL OCSUB ($1,$1,LPS)
   SUBROUTINE LPS ($,$,$)
   COMMON NENTRY,ITMSIZ,ITEM(80),NEA(13),NTEM(20)
  1 LABA(5),LABB(5),NBA(155),NBB(155),NL(20),NEB(20)
   DATA N1,N2,N3,N7 /0304,0305,0306,0312/
   DATA LABA /5,6HVACANT,0,0,0/
   DATA LABB /5,10HNOT VACANT,0,0/
   DATA NEB/-14,3,-31,4,-46,3,-63,2,-82,2,-96,1,
  A-115,1,-132,1,4*0/
   DATA NL /
  A42H   PSU     SERIAL    SEGMENT TYPE-QTRS     ,
  B42HVAC STAT SEAS STAT  USUAL OCC    COND      ,
  C36H                                          /
   IF (NENTRY) 10,20,30
10 RETURN 1
20 CALL OENFIL (4HOUTA)
   CALL OENFIL (4HOUTB)
   RETURN 1
30 IF (N.NE.0) GO TO 32
   LABA(2)=LFX(LABA(2))
   LABB(2)=LFX(LABB(2))
   LABB(3)=LFX(LABB(3))
   DO 31 J=1,20
31 NL(J)=LFX(NL(J))
   CALL INSERT (31,NL,1)
   CALL OPNOUT(4HOUTA,NBA,155,20,1,0,LABA,0,0)
   CALL OPNOUT(4HOUTB,NBB,155,20,2,0,LABB,0,0)
   CALL ITMOUT(4HOUTB,NL)
   NL(11)=0
   NL(12)=0
   NL(13)=0
   CALL ITMOUT(4HOUTA,NL)
   N=1
32 IF (ITEM(6).EQ.N1.OR.ITEM(6).EQ.N2.OR.
  A ITEM(6).EQ.N3) GO TO 40
33 CALL PACK (ITEM,NEB,NTEM)
   CALL ITMOUT(4HOUTB,NTEM)
   RETURN 3
40 IF (ITEM(8).NE.4) GO TO 33
   IF (ITEM(7).LT.N1.OR.ITEM(7).GT.N7)
  A GO TO 33
   CALL PACK (ITEM,NEB,NTEM)
   NTEM(11)=0
   NTEM(13)=0
   CALL ITMOUT (4HOUTA,NTEM)
   RETURN 3
   END
```

17.5. SPECIFICATIONS FOR MARRIAGE TABLE 7A

1. INPUT FILE: 89,171 27-word records
"=67 SEO EDITED-MARITAL"
IIIA Tape

2. ALL field values are in XS3.

3. First marriage date is recorded in field no. 7 for persons married more than once, and in field no. 5 for persons married only once.

4. Tally using weight in field 9. Weight is 6 digits with 2 implied decimals.

5. Output tables should show tally rounded to thousands.

6. Provide table title, stubs and captions as shown except for numbers 1-15 in left-hand margin of stub.

7. "TOTAL" for races includes "OTHER".

8. Survey date is 1967.

9. Tables should include only 14-69 year olds.

17.5. SPECIFICATIONS FOR MARRIAGE TABLE 7A (Continued)

10. INPUT FIELD LOCATIONS:

FIELD	WORD	DIGIT	UNIVERSE	DESCRIPTION	CODES
1	3	3-4	14-69 years	AGE	00-99
2	3	5	14+	MARITAL STATUS	
				MARRIED:	
				Spouse Present	0
				Absent (Armed Forces)	1
				Absent (Other)	2
				WIDOWED:	3
				DIVORCED:	
				Divorced	4
				Separated	5
				NEVER MARRIED	6
3	3	6	14+	RACE	
				WHITE	0
				NEGRO	1
				OTHER	2
4	4	1	14+	SEX	
				MALE	0
				FEMALE	1
5	4	4-5	EVER MARRIED 14+ (0-5 in MARITAL STATUS)	YEAR OF MOST RECENT MARRIAGE	00-99 /1
6	7	6	EVER MARRIED 14+	TIMES MARRIED:	
				Once	0
				Twice	1
				Three +	2
7	5	5-6	MARRIED MORE THAN ONCE	YEAR OF FIRST MARRIAGE	00-99 /1
8	6	1	MARRIED MORE THAN ONCE	CAUSE OF END OF FIRST MARRIAGE:	
				Death of Spouse	0
				Divorce	1
				Other	2
9	11	1-6	14+	PERSON'S WEIGHT	

/1 Anything above 67 is in 1800's.

Table 7a.--NUMBER OF TIMES MARRIED, CURRENT MARITAL STATUS, AGE AT FIRST MARRIAGE, HOW FIRST AND LAST MARRIAGE ENDED, AND YEARS SINCE FIRST MARRIAGE, FOR EVER MARRIED PERSONS: 1967

Subject	Total				White	Negro
	Total ever married	Years since first marriage			(Repeat as for Total)	(Repeat as for Total)
		Less than 10	10-19	20 and over		

MALE

Total.....

Married once
1 Married (at time of survey)
2 Widowed
3 Divorced

Married twice
Married
4 Widowed after first marriage
5 Divorced after first marriage
Widowed
6 Widowed after first marriage
7 Divorced after first marriage
Divorced
8 Widowed after first marriage
9 Divorced after first marriage

Married 3 or more times
Married
10 Widowed after first marriage
11 Divorced after first marriage
Widowed
12 Widowed after first marriage
13 Divorced after first marriage
Divorced
14 Widowed after first marriage
15 Divorced after first marriage

Repeat, same as Total, for:
14 to 21 years old at first marriage
22 to 27 years old at first marriage
28 or over at first marriage

FEMALE

Repeat, same as MALE, except change ages at first marriage to:
14 to 19
20 to 24
25 or over

PERCENT DISTRIBUTION

Repeat, showing vertical percentages for each of 8 blocks of lines on age at first marriage. Thus, the line for "Total" for males will have a line of 100.0's, and so will the total line for 14 to 21 years of age at first marriage, 22 to 27, and 28 or over. Same for females.

18. FORTRAN V

FORTRAN V

A number of features not found in FORTRAN IV have been included in FORTRAN V for use with EXEC 8 on the UNIVAC 1108. This chapter is devoted to these new features. Programmers should refer to the Programmers Reference Manual UP - 4144 for EXEC 8 components and concepts with which the FORTRAN V programmer should be familiar. A Census processor "FORT45" is available to translate a FORTRAN IV program into a FORTRAN V program which can be executed under EXEC 8. For information about EXEC 8 SORT/MERGE programs refer to the manual UP - 7621 and to the write-up "FORTRAN SORT LINKAGES".

In order to assemble a FORTRAN V program, there are various options associated with the EXEC 8 Processor Call Statement "FOR":

A. Accept the results of processing despite detected errors.

I. Insert. Introduce new source language element into Program File from the control stream.

L. Produce a complete printer listing.

N. Listing of only summary information.

S. Source code (and diagnostic message, if any) printout.

U. Update, produce new cycle of source language element.

W. List correction lines if corrections are applied.

X. Abort the Run if the Processor detects source language or compiler errors.

Note: The absence of a specific option related to printer output is treated as though the "N" option had been specified.

18.1. INCLUDE

The general form of the INCLUDE statement is

INCLUDE n, LIST

where n is the name of a PROC created by the Procedure Definition Processor. When "LIST" is used, the included statements will be listed whenever the source program is listed. The PDP element that contains the PROC must be in a file that is assigned to the run in which the compilation is being performed.

A normal use of this statement is the inclusion of the same specification, internal subroutines or DEFINE procedures in several programs.

18.2. DELETE

This statement provides a means to prevent the generation of object code following its appearance. The general forms of the DELETE are:

DELETE n

or

DELETE n, L

where n is a statement number where the generation of object code resumes. L is an integer constant or a PARAMETER variable with the value of 0 or 1. If L is 0, the statement is ignored; otherwise, it is executed as in the case of DELETE n.

18.3. EDIT

The EDIT statement allows the user to override the listing options on the "FOR" processor control card and to select the type of compilation listing produced for any portions of the source program. The word SOURCE refers to the source program statements and the word CODE refers to the octal and mnemonic representations of the generated code. The forms are:

1. STOP EDIT SOURCE

   This form terminates both source and octal code listings.

2. START EDIT CODE

   This form initiates both source and octal code listings.

3. STOP EDIT CODE

   This form terminates octal code listing only.

4. START EDIT SOURCE

   This form initiates source code only.

18.4. INPUT - OUTPUT END CLAUSE

FORTRAN V formatted read and write statements are the same as those of FORTRAN IV with the option of two new features - "END" and "ERR". The optional END Clause permits the programmer to retain control instead of terminating when an end of file is encountered.

18.5. INPUT - OUTPUT ERR CLAUSE

An ERR clause in a READ or WRITE statement suppresses program termination and allows retention of control when an error is encountered during execution of the statement. This is an optional clause. Forms for READ and WRITE statements are:

(1) READ (unit,3) list

(2) WRITE (unit,3) list

(3) READ (unit,z, ERR = l_1, END = l_2) list

(4) READ INPUT TAPE unit, f, list

(5) READ f, list

(6) WRITE (unit,z, ERR = l_1, END = l_2) list

(7) WRITE OUTPUT TAPE unit, f, list

(8) PRINT f, list

(9) PUNCH f, list

(10) READ (unit, ERR = l_1, END = l_2) list

(11) READ TAPE unit, list

(12) WRITE (unit, ERR = l_1, END = l_2) list

(13) WRITE TAPE unit, list

The various items used in forming the above statements are defined as follows:

unit - the logical unit involved

z - the format reference or a namelist name. If z is a namelist name, then the list of the statement must be void. (Refer to 18.18.)

list - an ordered list of variables, the contents of which are to be transmitted.

ERR = l - is an optional clause used to transfer control to statement label l which should be contained in the same routine as the input-output statement.

END = l - is an optional clause used to transfer control to statement label l when an end of file is encountered on input or the end of a mass storage file on output.

18.6. INPUT/OUTPUT STATUS WORD INSTAT FUNCTION

The content of the status word as set by a FORTRAN I/O statement may be retrieved for examination with the function INSTAT.

The status word has the following format:

Code	Error	Location
35 - 30	28 ----- 18	17 ------ 0

Code is expressed as an absolute integer. ERROR is either an integer or two left justified characters.

LOCATION is a program relative address. Codes and their meanings are listed on the following table.

Contents of the Input/Output Status Word

CODE (decimal)	ERROR	LOCATION	REMARKS
0	Format character/ internal type	Location of variable	Internal type and format type are incompatible
1	Error character	Location of format	Illegal format character
2	Maximum record length	Location of record	Maximum record length exceeded
3			Input/output ambiguity (possible hardware error)
4	Error character	Location of record	Illegal octal input
5		Location of record	Illegal integer input
6	Number of digits	Location of record	Too many digits in integer input
7		Location of record	Input exponent too large
8	Error character	Location of record	Illegal character in input
9		Location of table	Not namelist table
10	Relative position of variable name	Location of table	Illegal variable type
11		Location of table	File ended in namelist record
12	Error character	Location of record	Illegal character in namelist
13	Relative position of variable name	Location of record	Namelist input exponent too large
14		Location of record	Namelist input exponent error
15		Location of record	Namelist name of READ statement differs from that of input record

18.7. ERROR WALK BACK

An error occurring at any point in a subprogram causes a walk back to the main program. A message is printed at each subprogram level stating the name of the referencing subprogram, the sequence number of the reference, and the name of the subprogram referenced.

18.8. EXTERNAL SUBPROGRAM

An external subprogram may be written in FORTRAN V or in UNIVAC 1108 assembly Language. An external subprogram may be referenced from any part in the entire program. An actual argument may be an expression, an array, a subprogram name, or a statement number (preceded by $).

A FORTRAN element may also reference a COBOL subprogram by using the "CALL COBOL (entry name)" statement. Common working storage must be provided for creating or testing variables. The entry name used must be declared EXTERNAL.

A COBOL program may also call a FORTRAN subprogram by using the following statement:

"ENTER FORTRAN name SUBROUTINE REFERENCING variable$_1$ ...., variable$_n$"
Refer to UNIVAC 1106/1108 COBOL (UP-7626 rev.2).

18.9. MULTIPLE ENTRY POINTS

A provision is made for multiple entry points to a subprogram. These entry points may have their own names and argument lists including return arguments to specified statement numbers. The number and types of arguments are completely independent for each entry point; however, individual arguments set by one reference remain set for the entire subprogram until new arguments are supplied. Thus one entry could possibly initialize the sets, and other entries could provide those arguments which vary. The form of the ENTRY statement is:

ENTRY name (a_1, a_2, ..., a_n)

where name is the symbolic name of the entry point and can be considered an additional name of the subprogram, and a_i are the formal arguments.

References to an entry are made as to any subprogram. Functions return values and subroutines are called as usual. Example of a function reference:

A = name (a_1, a_2, ..., a_n)

Example of a subroutine call:

CALL name (a_1, a_2, lll, a_n)

Within the body of a subprogram, control flow ignores ENTRY statements.

The following example illustrates the use and advantages of multiple entry points. Initialization is done once with the first reference. Thereafter, the function is entered through an optional entry point.

```
C      AN EXCERPT FROM A REFERENCING PROGRAM
       V(1) = SCALAR (1,1,AA,BB,IT,KK,NN,1.0E-3,$5,$6)
       Do 4 KK = 2,NN
  4    V (KK) = FAST (1,KK)
       GO TO 7

  5    STOP 5
  6    STOP 6
  7    CONTINUE
C  A   SUBPROGRAM WITH ENTRY POINT FAST
C
       FUNCTION SCALAR (I,K,A,B,IMAX,N,TEST,$,$)
       DIMENSION A(IMAX,KMAX),B(KMAX,IMAX)
       IF (N.GT.IMAX,OR.N.GT.KMAX,OR.N.LT.1) RETURN 9
       DO 1 J=1,N
       IF (ABS(A(N,N)).LT.TEST.OR.ABS(B(N,N)).LT.TEST)RETURN 10
    1  CONTINUE
       ENTRY FAST (I,K)
       SCALAR = 0.
       DO 2 J = 1,N
    2  SCALAR = SCALAR + A(J,N)*B(N,K)
       RETURN
```

18.10. AUTOMATIC TYPING

For the intrinsic function (ABS, MOD, MAX, MIN, DIM, and SIGN) and for the mathematical library functions (SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SINH, COSH, TANH, EXP, LOG, LOG10, SQRT, and CBRT), the type of the function is determined by the type of the arguments. If arguments of differing type are used, they are combined according to the rules of mixed - type expressions and the function value is of the resulting type. Automatic typing also takes place when these functions are referenced by the alternate library names for specific type (for instance, the alternate library names for SIN are DSIN and CSIN) unless the compiler statement option (MATH = A) is used. This option inhibits automatic typing of the alternate names for mathematical library functions. If the name of a library function is passed as an argument, the type of the accessed function is determined by the type of the access function defined by name.

18.11. FLD FUNCTION and Bit Manipulation

The FLD intrinsic function extracts from or inserts into a 36 - bit string, any size byte specified. The typeless functions AND, OR, XOR, BOOL and COMPL manipulate bit strings. The form of the FLD reference is:

FLD (i, k, e)

where i is the beginning bit (left most bit of a word is bit 0) k is the number of bits (the width) and e is an expression of any type. The FLD function may be used in any arithmetic or relational expressions. The content of the byte extracted from the string is right adjusted in the arithmetic register and the remaining bits of the register are set to 0. The result is considered to be an integer value. The FLD function may appear to the left of the equals sign in an arithmetic expression. In this case, if the expression to the right is not integer, it is converted to integer. The right most bits of the value of the expression are inserted in the byte of the FLD's third argument as specified by the first and second arguments. The remainder of the word is not disturbed. Since the overall content of the third argument is changed, it must not be a constant, a PARAMETER valuable or an expression if the FLD function appears to the left of an equals sign. Examples of the FLD function:

Statement	Final Value
N1 = FLD (35, 1, 15)	N1 = 1
N2 =-FLD (33, 2, 15)	N2 = - 3
N3 = FLD (33, 1, 15) + FLD (34,2,15)	N3 = 1 + 3 = 4
N4 = FLD (35, 1, -1)	N4 = 0
N5 = FLD (34, 2, -1)	N5 = 2
N6 = FLD (1, 35, COMPL (0))	N6 = 2**35-1
A = FLD (1, 4, -1)	A = 15.0

(the integer result of FLD is converted to real)

Initial Value of N	FLD statement	Final value
0	FLD (34, 2, N)	N = 2
-4	FLD (34, 2, N) = 2.0	N = -1
PQRSTU	FLD (18, 6, N) = ABCDEF'	N = PQRFTU
1	FLD (0, 18, Z) = FLD (0,18,Z)	1 (no change)

Specified COMPILER statements may be requested to control the FLD interpretation and the generated code:

(FLD = R) specifies that bits are to be counted from right to left starting with bit 0.

(FLD = L) specifies counting from left to right

(FLD = Q) causes the compiler to generate optimized code for FLD references when the field specified is one of four quarter-words if the installation has set the quarter-word ability.

(FLD = T) causes optimized code for the three third-words if the FLD function is specified to the left of the equals sign and quarter-mode ability is present.

In summary, the FLD function permits the extraction of bits from a word and/or the insertion of bits into a specified portion of a word.

18.12. DEFINE

A Define procedure is referenced like a function with arguments. It must appear before any executable statements. In line code is generated with each reference. The form of the DEFINE procedure is:

DEFINE name (a_1, a_2,...., a_n) = expression

It is permissible to omit the word DEFINE if desired. Subscripting of subscripts is, in effect, permitted to any level through use of the DEFINE statement. For example:

```
DIMENSION B (100), I (100)

DEFINE BB (K) = B (K)

X = BB (I(J))
```

An example of a prodecure reference is:

```
DEFINE F (X, Y) = X + Y

A = F (U * V, V)
```

and the result is A = U * V + V

18.13. MIXED TYPE ARITHMETIC

Integer, real, double precision, complex and typeless expressions may be combined by the arithmetic operators including exponentiation. The only exception is that complex and double precision expressions must not be mixed with each other or

with typeless expressions. Conversion takes place when the type of the expression differs from the type of the receiving variable. Arithmetic expressions of differing types may also be compared in relational expressions.

18.14. 1108 DOUBLE PRECISION HARDWARE

The UNIVAC 1108 hardware allows efficient double precision arithmetic with a resolution to 18 decimal digits. The 1108 double-precision format contains an 11 bit characteristic biased by 1024 (2000g). The mantissa is composed of 60 bits extending into the second word.

S	characteristic		Mantissa	
71	70	60	59	36

35	Mantissa	0

18.15. PARAMETER VARIABLES

PARAMETER variables may be defined by expressions. A PARAMETER variable may not be redefined but it may be used in the definition of subsequent PARAMETER variables. For example:

```
      PARAMETER J = 1, ALPHA = 4 - J, K2 = ALPHA,
     A      K8 = 2 ** K2,
     B      K4 = (K8 - 5) /2 + K2/2 + 2
```

Following the PARAMETER statement, the use of J is equivalent to using the integer constant 1, and using 3 for ALPHA, 3 for K2, 8 for K8, and 4 for K4.

18.16. DATA IN TYPE AND DIMENSION STATEMENTS

It is now possible to include data with type and DIMENSION statements as well as in the DATA statement. However, implied DO loops are not allowed for type and DIMENSION statements. The following examples illustrate the different ways of including data:

```
Example 1:  DIMENSION R (7)
            DATA A, R /1.5, 7 * 3.5/
Example 2:  REAL A/1.5/ R (7)/7 * 3.5/
Example 3:  DIMENSION A (1)/1.5/R (7)/7 * 3.5/
```

Example 4: PARAMETER P = 3

INTEGER X/4/Y (P, 4), Z (P, 4,)/9, 2 * 5,

P * 6/ZZ

As a result of the above:

X = 4

Z (1, 1) = 9

Z (2, 1) = 5

Z (3, 1) = 5

Z (1, 2) = 6

Z (2, 2) = 6

Z (3, 2) = 6

Y and ZZ are undefined

Two COMPILER statements are available for DATA statement specifications:

(DATA = SHORT) allows the number of constants to be fewer than the size of an array when the list is unsubscripted.

For example, without the option, the following would result in a diagnostic "The list in the above statement is too long."

DIMENSION A (10)

DATA A / 1., 2., 3. /

(DATA = IBM) allows the insertion of more than 1 word of data into a DATA array when a subscripted reference is made. For example, without this option, the following usually results in a diagnostic:

DATA A (3) / 1, 3, 3 /

However with the DATA = IBM option, the action is to set 1 in A (3), 2 in A (4), 3 in A (5).

18.17. IMPLICIT

The IMPLICIT statement is used to specify that names starting with a certain letter are of a specified type. The IMPLICIT

statement overrides the name rule (where names I - N are typed integer and all other real) but is overruled by an explicit type statement.

The IMPLICIT statement may appear any place in a FORTRAN V program. The type associated with a letter is in effect until it is redefined by a following IMPLICIT statement. Once the type of a name is set, it cannot be changed. Local - global rules are followed for symbolic names.

The form of the IMPLICIT statement is:

IMPLICIT type (a_1, a_2, ...), type (a_3, a_4,...)
where a represents single alphabetic characters or a range of characters denoted by the first and last letters in the range separated by a hyphen. The letters are then associated with the type that precedes the parentheses. For example:

IMPLICIT INTEGER (A, X - Z), REAL (M) INTEGER M2 declares that names beginning with A, X, Y, Z are integer and that names beginning with M (except for M2) are real. The explicit typing may be used to define specific names. FORTRAN V allows for additional form for double precision typing. It is:

REAL * 8 $name_1$, $name_2$, ...

18.18. NAMELIST INPUT/OUTPUT

A list of subscripted or unsubscripted variables or arrays may be specified in a NAMELIST statement. When the name of this list appears in a READ or WRITE statement, the entire list of variables may be transmitted. Since no format is specified, standard editing codes are used depending on the type of the list elements. Input - output statements take the form:

READ (unit, X, ERR = l_1, END = l_2)

WRITE (unit, X, ERR = l_1, END = l_2)

where X is a namelist name. The contingency clauses ERR = 1, and END = l_2 are optional. The lists of items to be transmitted are void in the above statements, because the NAMELIST statement specifies the items to be transmitted. It is possible to write a simple statement with the same effect as a long list and a reference to a complicated FORMAT statement.

On input, the data to be read is identified within the input record itself. Thus, to read the number 1.5 into a variable A, the external input data field contains: A = 1.5 where fields are separated by commas. The type of the data should not conflict with the type of the list item as specified.

READ (unit, namelistname), the first character in each data record to be read is always ignored. The second character of the first record of a group of data records to be read must be a $ (or the character &) immediately followed by the namelist name and a blank. If the name is not the one declared in the READ statement, the input medium is searched for a match. The remainder of the first record with a namelist name and the following records may contain any combination of the legal data items which are separated by commas. The last input record is terminated by a blank followed by $END or &END. The constants used in the data items may be integer, real, double precision, complex, or logical. Logical and complex constants may be associated only with logical and complex variables, respectively. However, other types are converted in accordance with the type of the variable. Blanks may not be embedded in a constant or a repeat constant field, but may be used freely elsewhere. The name of an array and the value of its first elements must appear on the same record. The last item on each record that contains data items must be a constant followed by a comma. The comma is optional in the record that contains or precedes the $END sentinel.

Example of input data to be read using a namelist:

Column
1 2 3

Card 1 $ NAM1 I (2,3) = 5, 7, 9,

Card 2 0, 10, L (3) = 4.3, A = 4,3,

Card 3 8 * 4.3, J = 4.2

Last card $END

Example of a namelist and a READ statement to read above input data:

```
DIMENSION    A (10), I (5, 5), L (10)

NAMELIST/NAMI/A, B, I, J, L (3)

READ  (5, NAMI)
```

Result of above READ:

Integers 5, 7, and 9 are placed in I (2, 3), I (3, 3), and I (4, 3). Integers 0 and 10 are placed in I (5, 3) and I (1, 4). The real constant 4.3 is converted to integer 4

and placed in L (3). Integers 4 and 3 are converted to real constants and placed in A (1) and A (2). Real constant 4.3 is placed in A (3) ------A (10). Real constant 4.2 is converted to 4 and placed in J.

The list of items in the namelist is used on input to specify those items which may have their values defined in the records to be read. Not all items of the namelist need be used in the input records nor must the input fields be in the same order as the list items. If the list item is an array name, data may be assigned to the entire array or any individual elements as specified by the input records. On output, each list item of the namelist is formatted in a standard fashion for output in the order specified by the list.

The output data is written such that:

1. The name of a variable and its value are written on one line.

2. The name of an array is written, with the values of the elements of the array written in a convenient number of columns, in the order of the array in main storage, that is, with the left dimension.

3. The data fields are large enough to contain all the significant digits.

4. The output can be read by an input statement referencing the namelist name.

The form of the NAMELIST statement is:

```
NAMELIST / X / A, B, ... C/Y/D, E, ....F/Z/G, H, ....I
```

where X, Y, and Z are namelist names and A, B, C, D, E, F, G, H, I are simple variables, subscripted variables, or array names.

Example:

```
DIMENSION A (10), I (5, 5), L (10)

NAMELIST / NAM1 / A, B, I, J, L (3)/NAM2/ A, C, J, K, I, (2, 3)
```

In the preceding example, arrays A and I, variables B and J, and subscripted variable L (3) are associated with namelist name NAM1 and array A and variables C, J, K, and I (2, 3) are associated with namelist name NAM2.

Rules for defining and using a namelist:

1. A namelist name consists of from one to six alphanumeric characters, the first of which must be alphabetic.

2. Within a NAMELIST statement, a namelist name is enclosed in slashes. The list of variables associated with a namelist name ends when a new namelist name enclosed in slashes is encountered or with the end of the NAMELIST statement.

3. A namelist name may be defined only once in a routine by its appearance in a NAMELIST statement and it may be used only in input/output statements.

4. A namelist name has the same local-global properties as any variable.

5. A namelist name must be unique in the routine in which it appears.

6. A variable name or an array name may be assigned to one or more namelist names.

7. The subscript (S) of an array element must consist of constants or PARAMETER variables.

8. A variable in a NAMELIST statement may be a formal argument.

9. If a namelist element is an array, its maximum dimension may be transmitted by an argument list or by COMMON variables.

18.19. FORMAT SCAN POINTER T

The T editing code selects a given character position for formatted input/output, thereby allowing modification of the normal left to right scan of a record.

FORMAT (10X, F10.3) is equivalent to

FORMAT (T11, F10.3)

On output, the order of a list need not be the same as the printed page. For example:

FORMAT (T50, F10.3, T5, F10.2) is valid for printing the first list item in position 50-59 and the second list item in print position 5-14.

18.20. G EDITING CODE

This is a general editing code which provides for real, integer, double precision, and logical conversion depending upon the variable types. For real data, various actions occur. If the output is real, an F or E editing code is used depending on the absolute value of the item. When the F form is used, an 0 scale factor is assumed. For real input, the G editing code is the same as an F editing code without scale factor.

18.21. REREAD

An input/output unit number may be assigned for rereading of the last record read. Thus, it is possible to read the last record any number of times. Input items of a card read with editing codes according to their types could, for instance, be reread with an A-type editing code as alphanumeric information. Only subsequent formatted READ and ENCODE statements change the records to be reread. Unit 30 is presently assigned via the NTAB table.

18.22. FREE FORMAT INPUT/OUTPUT

For free-field format input, the two external fields are separated by a comma. The end of the input image is a field end. The format specification referenced by the READ statement must be empty. All types of variables may be read but no Hollerith or octal input is possible. A standard G-type editing code is used. For example:

```
READ (unit, 13) K, V, C (KK (N), N = 1, 7, 2)

13 FORMAT ( )
```

For free field format output, the WRITE statement references empty format specifications. All five variable types (except alphanumeric information) may be written. A standard G-type editing code is used. For example:

```
WRITE (unit, 31) J, B, D, (AR (M), M = 9, 2, -2)

31 FORMAT ( )
```

No more than six fields of 13 positions are written in free-field output. Complex variables require two fields, both of which must be written on the same record.

18.23. MASS STORAGE FILES

Mass storage files on drum or FASTRAND may be used interchangeably with tape files. This is a feature implemented in EXEC 8 via the ASG and USE control statements. Refer to the Programmer's Reference Manual UP - 4144.

18.24. ENCODE/DECODE

Core-to-core data transfers under format control can be made using ENCODE or DECODE statements. The standard format scanning and editing capabilities of FORTRAN V are utilized. In other words, data can be formatted without the use of I/O. The general forms of the statements are:

ENCODE (block, f) list

and

DECODE (block, f) list

In the above statements, list represents a standard input/output list, f is the format statement number or the name of an array containing format specifications, and block is the name of the array or variable to/from which data is to be transferred.

The execution of an ENCODE statement causes the contents of list to be converted according to the specified format and the results to be stored in block. This action is similar to a WRITE statement which generates Fieldata characters. Execution of the DECODE statement causes the contents (in Fieldata) of the block to be moved into list items in accordance with the format specification. This action can be likened to a READ statement except data is moved between areas of core storage.

18.25. OPTIMIZATION

The most outstanding feature of the UNIVAC 1108 FORTRAN V compiler is its optimization capability which results in extremely efficient coding. Numerous methods are employed: local optimization, interstatement optimization, common expression evaluation, register usage optimization, and inline code generation.

18.26. MASTER SPACE COMMENT

If a master space (7/8 punch or @ sign) occurs on a line from column 2-80, then any comment may be placed after the master space on the same line.

18.27. Ru FORMAT CODE

This code is similar to Au except that if u is less than 6, the rightmost characters of the list item are transmitted.

18.28 SUBSTITUTION FOR uH

If Hollerith characters are to be enclosed in a list, then the character may be enclosed in single quotes as a substitution

for uH. If a single quote is to be used as a character, then two quotes must be used for that character. Example: '''' is the same as 1H'.

18.29. COMPILER STATEMENT

Certain options can be declared with this statement to be effective for the entire programs unit. The form of the COMPILER statements:

COMPILER ($option_1$), ($option_2$), ..., ($option_n$)

Some of these options have been defined prior to this section. A summary of these options follows:

(FLD = L) count bits left to right (35-0)

(FLD = R) count bits right to left (0-35)

(FLD = Q) quarter word optimization

(FLD = T) third word optimization

(DATA = SHORT) allows the number of constants to be fewer than the number of variables provided that the list of variables is an unsubscripted array

(DATA = IBM) allows variables to be placed in consecutive locations of an array although 1 is named in the DATA statement.

(MATH = A) used to remove the names of all library functions other than generic names from the list of automatically typed function to prevent sympol table overflow.

(ADR = IND) used to generate code for indirect referencing of subprogram arguments to reduce code generated for multiple references to an argument. If used, both referenced and referencing routines should be compiled under their option. The code generated under this option produces subprograms that are re-entrant with respect to subprogram entry, exit, and argument retrieval. Under control of this option, the argument's address is retrieved and stored in NTEMP$ + i. All references to the argument are accomplished by addressing NTEMP$i indirectly.

18.30. MATH-PACK AND STAT-PACK

Of interest to mathematicians and statisticians are the extensive and comprehensive mathematical and statistical library subprograms which are described in Program Abstracts MATH-PACK (UP-4051) and STAT-PACK (UP-4041). This library provides the more frequently used tools of numerical and statistical analysis in order to speed up the preparation of solutions to many problems encountered in scientific research.

MATH-STAT subprograms fall into the following categories:

- Interpolation
- Numerical Integration
- Solution of Equations
- Differentiation
- Polynomial Manipulation
- Matrix Manipulation
- Differential Equations
- Systems of Equations
- Curve Fitting
- Random Number Generators
- Specific Functions

STAT-PACK provides the following types of subprograms:

- Descriptive Analysis
- Elementary Population Statistics
- Distribution Fitting and Plotting
- Chi-Square Tests
- Confidence Intervals
- Analysis of Variance
- Regression Analysis
- Time Series Analysis
- Multivariate Analysis
- Distribution Functions
- Inverse Distribution Functions
- Miscellaneous

19. REFERENCE INFORMATION

19.1. SOURCE PROGRAM ORDERING

The following ordering of statements is suggested. Deviation from these rules is not necessarily wrong; however, their observance may prevent some common programing errors.

1. Program declaration statement.

   a. In a main program, there is no program declaration statement.

   b. In a subprogram there must be a statement of either:

      (1) type FUNCTION F (a_1, a_2, ..., a_n)

      or

      (2) SUBROUTINE S (a_1, a_2, ..., a_n)

2. PARAMETER Statements.

3. Type Statements.

4. DIMENSION and COMMON Statements.

5. EQUIVALENCE Statements.

6. DATA Statements (in any event, must appear before use).

7. FORMAT Statements (may appear anywhere throughout a program).

8. Statement Functions (must appear before the first executable statement of a program).

9. Executable Statements.

10. The END Statement (at the end of a program unit only).

19.2. LIMITATIONS

Maximum statement number	32767
Maximum continuation cards	19
Maximum number of digits INTEGER	11
Maximum magnitude INTEGER	2^{35-1}
Maximum number of digits REAL	9
Maximum DOUBLE PRECISION	17
Magnitude DOUBLE PRECISION	10^{38}
Maximum characters in a name	6
Maximum array dimension	7

19.3. OCTAL-DECIMAL INTEGER CONVERSION TABLE

0000 to 0777 (Octal)	0000 to 0511 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal)	0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0672	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

19.3. OCTAL-DECIMAL INTEGER CONVERSION TABLE (Continued)

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

Octal	Decimal
2000 to 2777 (Octal)	1024 to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

Octal	Decimal
3000 to 3777 (Octal)	1536 to 2047 (Decimal)

19.3. OCTAL-DECIMAL INTEGER CONVERSTION TABLE (Continued)

OCTAL-DECIMAL INTEGER CONVERSION TABLE

4000	2048
to	to
4777	2559
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000	2560
to	to
5777	3071
(Octal)	(Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

19.3. OCTAL-DECIMAL INTEGER CONVERSION TABLE (Continued)

OCTAL-DECIMAL INTEGER CONVERSION TABLE

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

6000 to 6777 (Octal)	3072 to 3583 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

7000 to 7777 (Octal)	3584 to 4095 (Decimal)

19.4. 1107 OPERATOR TERMINATION CODES

CATEGORY	TERMINATION CAUSE	CODE
PD	Program/Data Troubles:	
	1. Program communication of inability to continue, or in loop -- terminated with dump (Use also for reruns terminated w/o dump if original run was terminated with a dump)	PD 1
	2. Other program/data incompatability (excluding setup or operating instruction problems)	PD 2
SE	Setup Troubles:	
	1. Program/inputs missing or incorrect	SE 1
	2. Operating instructions missing or incompatable	SE 2
	3. C/T cards incorrectly stacked	SE 3
HE	Human Errors:	
	1. Technician/Operations personnel error	HE 1
	2. Scheduling error	HE 2
TR	Tape Read: All tape read difficulties excepting those due to hardware or physical tape defects (x = last four digits of tape number)	TR(xxxx)
FL	Hardward Malfunction: All hardware failures except Servo hardware malfunctions (x = channel I.D. -- for central computer, type in "Z")	FL(xx)
SF	Servo Malfunction: All servo hardware failures (xx = Servo I.D.)	SF(xx)
TB	Tape Troubles: All physical tape defects including dirty tapes (x = last four digits of tape number)	TB(xxxx)
OT	Miscellaneous:	
	1. Function end, or to expedite schedule	OT 1
	2. Termination at request of sponsor	OT 2

19.5. ERROR CODES

LION FILE CONTROL ERROR EXIT

19.5.1. LION ERROR CODES THAT INDICATE EQUIPMENT MALFUNCTION

3A Uniservo

32 In an end-of-reel operation, a request to EXEC I for a rewind
33 (code 32) or a rewind with Interlock (Code 33) results in an error status. This indicates either equipment trouble or that the tape unit is otherwise engaged. No recovery.

37 When opening an output file, the preliminary rewind returned an error status signal from EXEC I. This indicates either equipment trouble or that the tape was otherwise engaged. No recovery.

38 When opening an output file, a read backward operation during the preliminary positioning of the tape returned an error status signal from EXEC I. This indicates either equipment trouble or that the tape unit was otherwise engaged. No recovery.

39 To reposition the tape for a rewrite of a block which has just been written or read in a forward direction, the tape must be moved back two blocks and then forward one block. The move forward is not completed correctly, and an error status signal is received from EXEC I. This is almost invariably caused by equipment trouble No recovery.

40 In attempting to reposition the tape for overwriting, a backward read has not been successfully completed. No recovery.

41 Failure to read a block after successful repositioning described in error code 40. No recovery.

42 Incorrect positioning of a tape at the beginning of a reel, probably due to equipment malfunction. No recovery.

43
44 Incorrect positioning of an input tape at a label block or
45 End-of-File sentinel. No recovery.

2A Uniservo

108 The rewind after the typeout following a checksum operation has not been completed normally. No recovery.

114 After a write error, an attempt to move the tape backward has indicated another error, other than an end-of-file. No recovery.
115 Tape positioning error in a write operation. No recovery.

19.5.2. LION ERROR CODES

CODE Octal	CODE Decimal	USE	ORIGIN	DESCRIPTION
1	1	IO EO	IODRUP	The output file has filled up the available drum table area to a point where another full buffer cannot be written. The last item presented for writing has not been recorded on the drum. The file is left in an open condition. For Internal Format files, the sentinel word will have been recorded, and the drum file can therefore be used successfully as an input file. Recovery may possibly be effected by storing zero in the lower half of pa + 16, lengthening the amount of usable drum area at pa + 20, and re-executing the calling sequence which caused the error. To close the file, a non-zero value should be stored in bits 29-24 of pa + 12. Note that if LION had attempted to write in a drum area not assigned to the program, the jump would have been to the Write Error Exit.
2	2	II	IO165A	The input buffer size is smaller than the block size as recorded on tape or drum. Recovery may possibly be effected by setting bits 11-6 of pa - 26 to a non-zero value, increasing the buffer size, and reopening the input file. Position has not been lost on the input tape or drum.
3	3	II IO EI EO	IOENDF	While closing a file, or a reel, the rewind parameter was a six-letter word not ending in D, T, or L. LION tests the last letter of REWIND, REWINT, or NOREWI to determine the rewind procedure. Any left justified coding of less than six characters will have the same effect as REWINT. To recover, change the code in the appropriate IBANK macro (e.g., at OENREL + 3), and re-execute the close file or close reel calling sequence.
4	4	IO	OPNOUT	An attempt to open an already open output file has been made. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
5	5	IO	IOKBEG	An attempt to perform an ITMOUT, BYPASS, OENFIL or OENREL operation has been made, with the file either closed or demanding an OENREL. No recovery.
6	6	II	ITMIN	An attempt has been made to perform an ITMIN operation with the file not yet opened, or when an End-of-File or End-of-Reel calling sequence is the only legitimate operation. No recovery.
7	7	II	IENREL	An attempt has been made to execute an End-of-Reel procedure on an already closed input file. No recovery.
10	8	II	IENFIL	An attempt has been made to execute an End-of-File procedure on an already closed input file. No recovery.
11	9	II	OPNIN	An attempt has been made to open an already open input file. No recovery.
12	10	IO	OPNOUT	When opening an output file, LION has found that the fixed item size is greater than the buffer size. Recovery may be possible by modifying the buffer size in the upper half of pp + 19, setting bits 29-24 of pa + 12 to a non-zero value, and reopening the file. Or, if the buffer had been coded symbolically, it may be increased via the TAL card and the program can be rerun.
13	11	IO	ITMOUT	An Item in a variable item size file is too large for the output buffer. Recovery procedures are similar to code 12_8.
14	12	IO	IO014C	During the preliminary positioning of an output file, a read backward operation has detected a block which is neither a label block nor an End-of-File sentinel block; indicating that the tape had been improperly positioned by the user prior to opening the file. Normally no recovery is possible.
15	13	II EI	IO165A	There is an invalid flag in bits 35-33 of the label block, indicating that probably this tape was not originally written by LION. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
16	14	IO	IO017A	As LION prepared to record a label block, it is found that the size of the File Label (including Free Words) is greater than the maximum allowed for the specified block size, i.e., 113 words for a 120 word tape block, 233 words for a 240 word tape block, and 5 words less than the buffer size for drum files. No recovery.
17	15	IO EO	IOTBEG	The output file being referenced does not have a tape unit or drum table assigned. No recovery.
20	16	IO	IOTBEG	The buffer size of a tape output file is less than 120 words. Recovery is possible by changing the buffer size in the upper half of pa - 19, setting the close file flag by placing a nonzero value in bits 29-24 of pa - 12, and rerunning.
21	17	IO	IOTBEG	No output buffer address has been specified. No recovery.
22	18	II EI	IO1134	An attempt has been made to continue writing at a point other than the end of an existing file, with block size less than 240 words. No recovery.
23	19		IODRU3	The buffer size specified is smaller than the block read from drum.
24	20	II	IOI87A	The first block of the file is not a label or sentinel block, since it does not contain the sentinel word 757574757475. The tape has not been properly positioned, and recovery is not recommended. However, setting bits 11-6 of pa - 26 to a nonzero value and programing a LOCFIL may in some cases allow continued processsing.
25	21	II EI	IOI65	In a read forward operation a 50-word sentinel block has been read, where a label block of 120 or 240 words is expected. See code 24_8 for possible recovery.
26	22	II EI	IOI65	The first block in a file is not a label block in the LION format, since the flag in bits 35-33 of the second word is equal to zero. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
27	23	II	IODATA	This indicates a checksum error. A recomputation from the data read in does not agree with the checksum originally recorded. This normally indicates equipment malfunction, and recovery is not feasible.
30 31 32 33	24 25 26 27	II EI	IOSEND	Each of these error indications results from detection of an invalid flag in a sentinel or label block. No recovery.
34	28		IODRU3	The buffer size specified is smaller than the block read from drum.
35	29	II	IOI42A	Opening file backward on UNISERVO IIIC illegal. No recovery.
40 41	32 33	IO EO	ITMOUT (IO07) (IO012)	In an End-of-Reel operation, a request to EXEC I for a rewind (code 32) or a rewind with interlock (code 33) results in an error status. This indicates either equipment trouble or that the tape unit is otherwise engaged. No recovery.
42	34	IO	IOCHDR	The file which is to be written has a buffer length less than 240, and its label block is 120 words in length. This file can be written only at the beginning of a tape, or following the last existing file on the tape. The tape is not properly positioned to meet these requirements. No recovery.
43	35	IO EO	IOCHDR (IO014D)	When opening an output file, the tape is positioned at a sentinel block, but it is neither an End-of-File sentinel nor the label block of the first file on the tape. No recovery.
44	36	IO EO	IOCHDR (IO014D)	In attempting to write a file which is to follow any existing files, a read of the last block indicates that it is not an End-of-File sentinel block. The tape is therefore improperly positioned. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
45	37	IO EO	IOCHDR (IO014A)	When opening an output file, the preliminary rewind returned an error status signal from EXEC I. This indicates either equipment trouble or that the tape unit was otherwise engaged. No recovery.
46	38	IO	IOCHDR	When opening an output file, a read backward operation during the preliminary positioning of the tape returned an error status signal from EXEC I. This indicates either equipment trouble or that the tape unit was otherwise engaged. No recovery.
47	39	IO EO	IOREPO	To reposition the tape for a rewrite of a block which has just been written or read in a forward direction, the tape must be moved back two blocks and then forward one block. The move is received from EXEC I. This is almost invariably caused by equipment trouble. No recovery.
50	40	IO EO	IOREPO	In attempting to reposition the tape for overwriting, a backward read has not been successfully completed. No recovery.
51	41	IO	IOREPO	Failure to read a block after successful repositioning described in error code 40. No recovery.
52	42	IO EO	IOREPO	Incorrect positioning of a tape at the beginning of a reel, probably due to equipment malfunction. No recovery.
53	43	IO EO	IORHDR	Incorrect positioning of an output tape at a label block or End-of-File sentinel. No recovery.
54	44	IO EO	IORHDR	Incorrect positioning of an input tape at a block which is not a label or EOF sentinel block. No recovery.
55	45	II EI	IO187A	While positioning the tape for a forward read operation, an error status code has been received from EXEC I. This is due either to incorrect positioning or equipment malfunction. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
56	46	II	IO175	During the search for a second BYPASS sentinel, an error status code has been received from EXEC I. No recovery.
57	47	II	CONWRI	During the positioning of a tape for the continue write function, an error status code has been received from EXEC I. No recovery.
60	48	II	IENFIL	The attempt to terminate all outstanding requests on this file has resulted in an error status. No recovery.
61	49	IO	CONWRI	Omission of parameter cc in the CONWRI macro specifies that the tape should be positioned at a label block or EOF sentinel block. The error code indicates that the tape is not so positioned. No recovery.
62	50	IO	CONWRI	A backward read operation in the positioning for a continue write results in an error status. No recovery.
63	51	IO	CONWRI	Same as code 50 except that a forward read caused the error. No recovery.
64	52	EO	IMGOUT IOXITM IOPP2	An attempt to punch a card or print a line resulted in an error status signal. No recovery.
65	53	IO EO	OENFIL IOPP3	An attempt to punch any one of the four stop cards resulted in an error status signal. No recovery.
66	54	EI	IOPP4	With input on punched cards, an attempt to condition the card read for Fieldata input has resulted in an error status signal. No recovery.
67	55	EI	IOPP4	With input on punched cards, an attempt to read a label card has resulted in an error status signal. No recovery.
70	56	EI	IOTES2	In attempting to type out, "LABEL CARD WASN'T LOADED PROPERLY," an error status signal was received from EXEC I. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE Octal	CODE Decimal	USE	ORIGIN	DESCRIPTION
72	58	IO EO	IOCHDR IO014B	During the process of repositioning the tape after reading the last block recorded in any preceding file, an attempt to perform a contingency write operation on a UNISERVO IIIA tape unit results in an error status signal. No recovery.
74 75 76	60 61 62	IO	IO03CC	Each of these errors occurs when LION expects to read a sentinel block or label on a IIIC prior to writing the file. No recovery.
100	64	EO	IMGOUT	An IMGOUT has been requested on an output file that has not been opened. No recovery.
101	65	EI	IMGIN	An IMGIN has been requested on an input file that has not been opened. No recovery.
102	66	EO	IMGOUT IOTAD	The a-field of an output operation parameter is zero. No recovery.
103	67	EO	IMGOUT	The a-field of an output operation parameter is greater than 9. No recovery.
104	68	EO	IMGOUT IOTAU6	The number of output characters for a floating point binary to fixed point decimal conversion is greater than ten. No recovery.
105	69	EO	IMGOUT IOTAU6	In a floating point binary to fixed point decimal conversion, the number of digits to the right of the decimal point is greater than the output character count minus 2. No recovery.
106	70	EO	IMGOUT IOTAV	In a floating point binary fixed point decimal conversion, the number of character positions reserved for the integer portion of the converted number is not large enough to contain the generated integer. No recovery.
107 110	71 72	EO	IMGOUT IOTAR	The floating point number is too great (code 71) or too small (code 72) for floating point binary to floating point decimal conversion. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
111	73	EO	IOTAFA	The attempted execution of an output parameter operation has caused the total number of characters going into the output image to exceed the image size specified by parameter pt of the EOCON macro. No recovery.
112	74	EO	IMGOUT IOTB06	In a fixed point binary to fixed point decimal conversion, the number of output characters is greater than 12. No recovery.
113	75	EO	IMGOUT IOTB06	In a fixed point binary to fixed point decimal conversion, the number of places specified to move the decimal point to the left, after conversion, is greater than the number of characters which are to be written. No recovery.
114	76	EO	IOTBS2	In a fixed point binary to fixed point decimal conversion, the number of character positions reserved for the integer portion of the converted number is not large enough to contain the generated integer. No recovery.
115	77	EI	IONAFA	The attempted execution of an input parameter operation has caused the total number of characters read from the input image to exceed the image size determined by the input medium specified by parameters pi and pk of the EICON macro. No recovery.
116	78	EI	IMGIN IONAP	An input character is not a valid Fieldata number. No recovery.
117	79	EI	IMGIN IONAR	The exponent of a floating point number does not fall within the range of -38 to + 39. No recovery.
120	80	EI	IMGIN IONAV2	The number of places that the decimal point is to be moved to the right, before conversion, has exceeded the limit 10-I. No recovery.
121	81	EI	IMGIN IONAY	In a fixed point decimal to fixed point binary conversion, the number of bit positions reserved for the integer portion of the converted number is not great enough to contain the generated integer. No recovery.

19.5.2. LION ERROR CODES (Continued)

CODE		USE	ORIGIN	DESCRIPTION
Octal	Decimal			
122	82	EI	IONTES	The first input character of a number which is to be converted is neither a sign nor a space. No recovery.
123	83	EI	IONAL IONAM	The number of input characters for any of the following conversions is too great: floating point to floating point, floating point to fixed point, or fixed point to fixed point.
124	84	EI	IONBCA	The octal character for either the octal or BCD transfer is incorrect.

THE FOLLOWING ERRORS OCCUR ONLY IN FILES USING THE FIXED BLOCK UNISERVO IIA TAPE FORMAT:

140	96	IO	OPNOUT	An attempt has been made to open an already open output file. No recovery.
141	97	IO	OPNOUT	The label image for an output file is too long. No recovery.
142	98	IO	ITMOUT	An output item has been presented to a closed file, or to a file demanding a close file calling sequence. No recovery.
143	99	IO	OENFIL	An attempt has been made to close an already closed output file. No recovery.
144	100	IO	OENFIL	A backward read operation in closing a file has not been completed normally. No recovery.
145	101	II	OPNIN	An attempt has been made to open an already open input file. No recovery.
146	102	II	OPNIN	An error has been detected in the initial forward read of a label block. No recovery.
147	103	II	ITMIN	An input item has been requested from a closed file, or from a file demanding a close file calling sequence. No recovery.
151	105	II	IENFIL	An attempt has been made to close an already closed input file. No recovery.

19.5.2. LION ERROR CODES (Continued) FIXED BLOCK UNISERVO IIA

CODE		ORIGIN		DESCRIPTION
Octal	Decimal			
152	106	IO	ITMOUT	A backward read in checksumming an output tape has not been successfully completed. No recovery.
153	107	IO	ITMOUT	A backward read of the last data block in checksumming an output tape has been successfully completed. No recovery.
154	108	II IO	ITMIN ITMOUT	The rewind after the typeout following a checksum operation has not been completed normally. No recovery.
155	109	II	ITMIN	The check read of a tape has not been completed normally. No recovery.
156	110	IO	ITMOUT	A backward read of the label block in checksumming an output tape has not been successfully completed. No recovery
160	112	II IO	IOHSKP	An incorrect version of LION has been called for. No recovery.
161	113	II IO	IOHSKP	The item size for the file is not a factor of 120. No recovery.
162	114	IO	ITMOUT	After a write error, an attempt to move the tape backward has indicated another error, other than an end-of-file. No recovery.
163	115	IO	ITMOUT	Tape positioning error in a write operation. No recovery.
166	118	IO-II		The file data area being requested is occupied by a previously opened file.
167	119	IO-II		The number of opened files exceeds 20.
170	120	IO-II		When opening a file one of the required arguments was not present.
171	121	IO-II		The peripheral equipment definition is not a legal value.
172	122	IO-II		In an item read or write, a close reel or file, or an ITMRTN statement, the file data area requested is not defined.

19.5.3. JOB REQUEST ERROR CODES

An error in the job request format will cause one of the following messages to be typed on the console printer in one of the following forms:

SP∆*JRID*∆ERnn or INT∆SP*JRID*∆ERnn

The error codes, ERnn, and their description are as follows:

ER1	Job request identity of current job too long. Message shows previous job request ID
ER2	Card type field too long
ER3	Illegal card type
ER4	PM cards out of order
ER5	TAL name too long
ER6	TAL number too long or illegal character on the TAL card
ER7	Illegal character on TRN card
ER8	TRN field too long
ER9	FAC name too long
ER10	Reserve on more than one card
ER11	Illegal facility requested
ER12	Illegal character on FAC
ER13	Illegal sequence on FAC
ER14	FAC asked for too large a core block
ER15	Two FAC card FACILITY REQUIREMENT fields contain asterisks
ER16	Job program name too long
ER17	Illegal character or sequence on PTY
ER18	Illegal number of characters in some field on PTY (Type of Run is indicated as REAL-TIME)
ER19	Rerun but no absolute units on PTY
ER20	Card count on PTY and cards in job do not match
ER21	PTY cards out of grouping
ER22	FAC cards out of grouping

19.5.3. JOB REQUEST ERROR CODES (Continued)

ER23	TAL cards out of grouping
ER24	TRN cards out of grouping
ER25	First card of new Jrid is not a PTY card
ER26	Duplicate request ID
ER27	No IBANK specified on FAC card
ER28	Illegal RUSH field on the PTY card or FAC card is not the second card of the job request
ER29	Job request core storage table filled
ER30	PM number is not a numeric
ER31	Matched priority and precedence of a job in schedule but no sequence number on new job
ER32	Matched priority and precedence of a non-sequence job
ER33	Priority and precedence of Z63 but not a permanent job
ER34	FAC card ended with colon but next card is not FAC and starting with a tag field
ER35	Configuration cannot satisfy FAC request
ER36	"Program Medium" or "Library Medium" field of PTY is not part of configuration
ER37	Period at an illegal point on any card of the job request
ER38	Library input tape symbol of "input Medium" field on PTY card equated to a program symbol and TRN card specifies the same program symbol as a unit to be transferred to another job
ER39	SUB cards not grouped together
ER40	SUB card field has new name but no old name
ER41	Subroutine name or entry point shown on SUB card is longer than six characters
ER42	Illegal absolute channel and unit in AB field on FAC card

19.5.3. JOB REQUEST ERROR CODES (Continued)

ER43 None, or more than one, unit tag in AB field on FAC card

ER45 *JRID* (PTY full)

ER46 *JRID* (REQT full)

ER47 PM. card missing

19.5.4. LOAD ERROR CODES

Error or warning conditions other than those connected with insufficient facility requests may be detected during the loading process. When an error or warning condition occurs, the operator is informed by a message on the console printer of the form:

LD *jrid* ER nn or LD *jrid* WAR nn

where nn is a two-digit error or warning code. The ER message causes loading to terminate, but the WAR message does not.

The warning or error codes follow:

00	Error in format or read of ROC program file
01	First block of program file not a label block
02	Checksum error on reading program file
03	Attempted to assign a segment storage drum table via a transfer, or to transfer the assignment of a segment drum table to a succeeding program
04	Not EXEC ROC program
05	Table length modification not allowed
06	Jump switch cannot be deleted
07	I/O facility cannot be deleted
10	Illegal systems reference in main program
11	Nonrecoverable I/O condition
12	Machine or peripheral equipment error
13	Program file does not contain symbol of drum table being transferred to program
14	Program file does not contain symbol of drum table requested for transfer to a succeeding program
15	Program file does not contain symbol of I/O facility being transferred to program
16	Program file does not contain symbol of I/O facility requested for transfer to a succeeding program
17	I/O facility requested for transfer to a succeeding program was deleted
20	Table (DALL) containing drum table assignment full, cannot transfer drum table to succeeding program
21	Requested DBANK area too small
22	Requested IBANK area too small
23	$PARAM table too small
24	Segment storage tape assigned to same unit as main or subroutine program file
25	No DBANK requested for load of a complex program
26	Subroutine requested segment storage
27	Tape from which subroutines are to be loaded is not a library tape
30	Library tape format error
31	Library directory too large for 2K buffer supplied by Loader
32	Subroutine or external reference not in library directory

19.5.4. LOAD ERROR CODES (Continued)

33	Total drum area requested not enough to include segment storage
34	Attempted to transfer more than one I/O facility or drum table to same symbolic reference in program
35	External reference or subroutine not found in subroutine library
36	Subroutine medium not defined
37	Subroutine jump switch not defined in main program
40	Unable to assign subroutine I/O facility
41	Illegal systems reference in subroutine
42	Subroutine drum or core table not defined
43	Subroutine drum or core table longer than length of corresponding table of main program
44	Subroutine IBANK or DBANK longer than length given in INFO block of library
45	Warning, unable to make all table length changes specified on TAL card
46	Segment storage medium deleted
47	Attempted to transfer to an independent drum table, a drum area less than the area specified in a dependent table (a dependent table is one which has the same starting address as an independent one)
50	Core table length increment on TAL card too large (16-bit sum is less than original length)
51	Requested $ERROR table length greater than length fixed by EXEC (97)
52	The segmented main program cannot be loaded from the subroutine library tape
53	Insufficient drum available to copy the main program from the subroutine library tape
54	Insufficient number of units requested on the logical channel zero
55	Assignment of an absolute facility changed from available between the time of selection and loading
56	Equipment type in modification record differs from equipment type of an absolute facility or a facility being transferred in
57	No subroutine medium defined and not all subroutines are being deleted by a SUB card
60	Too many entry/exit points to subroutine
77	Tape-to-drum copying of a library is using a block of core which had tentatively been assigned for the loading of a selected program. Before replying to the CPY-MORE, the suspended program may be selected in order to re-initiate the loading of that particular program. Also may indicate cross bank loading.

19.5.5. EXEC I/O ERROR MESSAGES

Format: I/O ERR *mm*-Ccc-Uuu-Du-Aaaaaaaaa-Fff-Ffff-Gff-Xii-Txxyyzz-b

b- N706024
N706054
N004445

All fields which would appear as shown above: usually only 3 or 4 fields are present in an I/O error typeout. If the message originates from the I/O Error Logging, ER is replaced with LOG. (see below)

mm - message number indicative of the condition of the error
Ccc - channel (decimal)
Uuu - unit number (decimal)
Du - drum unit number
Aaaaaaaaa - drum address (octal)
Fff - most recent external function transferred (Ffff for Uniservo IV)
Xii - interrupt code
Gff - nonsense interrupt (message # 15)
ii(from Xii) ii - 01 input monitor
ii - 02 output monitor
ii - 03 function monitor
all other - except interrupt code

Txxyyzz - error count for the tape which has just rewound on the channel and unit shown.

xx - 70 EI Counts - IIA, IIIA
yy - 60 EI Counts - IIA, IIIA
yy - 44 (READ) EI - IVC
zz - 24 EI Counts (IIA)
zz - 54 EI Counts (IIIA) - write only
zz - 44 EI Counts (IVC) - write only

N706024 - IIA
N706054 - IIIA
N004445 - IVC

The interrupt codes shown in the message have reached the limit for the tape UNIT shown. Message format is for the read error count, and the 45 is for the write error count.

External Interrupt Codes Displayed by I/O Error Messages:

Magnetic Tape
20 - Sequence error; CS
24 - Sequence error (two block read); CU
30 - Character-count; CS
50 - Illegal function (34 code on IVC)
54 - Tape unit select error; Write error (IVC)
60 - Parity error; CU
70 - Character-count; CU
74 - Interlock
44 - Write error (IVC)

19.5.5. EXEC I/O ERROR MESSAGES (Continued)

External Interrupt Codes Displayed by I/O Error Messages (Continued)

Magnetic Drum
- 06 - Parity error in overflow address
- 07 - Parity error not during continuous read
- 14 - Write fault
- 20 - CS sequence error
- 30 - CS character-count error
- 50 - Illegal function
- 60 - CU sequence error
- 64 - Parity error during continuous read
- 70 - CU character-count error
- 54 - Illegal address - either program or machine produced

Cards
- 20 - CS sequence error
- 30 - CS character-count error
- 50 - Illegal function
- 54 - Read/punch verification error
- 70 - Illegal character
- 74 - Interlock

High Speed Printer
- 50 - Illegal function
- 54 - Unit Select error
- 74 - Interlock

Paper Tape
- 01 - Punch compare error
- 02 - End of paper tape or tape motion error

Function Codes Displayed by I/O Error Messages:

Magnetic Tape
IIA (fixed block mode)
- Read Tape Forward - 52
- Read Tape Backward - 72
- Search Tape Forward - 56
- Search Tape Backward - 76
- Read Forward with Sentinel Check - 53
- Read Backward with Sentinel Check - 73
- Move Tape Forward - 51
- Move Tape Backward - 71
- Write Tape at Low Density - 03
- Write Tape at Low Density - 04
- Rewind Tape - 20
- Rewind Tape with Interlock - 21

19.5.5. EXEC I/O ERROR MESSAGES (Continued)

Function Codes Displayed by I/O Error Messages: (Continued)
Magnetic Tape (Continued)
IIIA (with interrupt)
- Write one block 12.5 KC - 11
- Write one block 25 KC - 12
- Read Forward one block (low) - 51
- Read Forward one block (Norm) - 52
- Read Forward one block (High) - 53
- Read Backward one block (low) - 71
- Read Backward one block (Norm) - 72
- Read Backward one block (High) - 73
- Search Read (low) - 55
- Search Read (Norm) - 56
- Search Read (High) - 57
- Search Read Backward (low) - 75
- Search Read Backward (Norm) - 76
- Search Read Backward (High) - 77
- Rewind - 30
- Rewind & Interlock - 31
- Terminate - 33
- Bootstrap - 50

IVC	EXEC - Hardware		
	200p.p.i.	556p.p.i.	800 p.p.i.
Write Binary	01-130	02-120	65-120
Write BCD	05-132	06-122	71-122
Write BCD/Translate	27-132	30-122	75-122
Write EOF	03-133	04-123	70-123
Read Binary	37-530	42-520	66-520
Read Binary/Sentinels	40-530	43-520	60-520
Search Read Binary	44-570	46-560	74-560
Read BCD	47-532	52-522	72-522
Read BCD/Sentinels	50-532	53-522	62-522
Search Read BCD	54-572	56-562	67-562
Read BCD/Translate	31-532	33-522	73-522
Read BCD/Trans/Sent	32-532	34-522	63-522
Search Read BCD/Trans	35-572	36-562	76-562
Backspace Block	61-303		
Backspace File	64-313		
Rewind	20-301		
Rewind/Interlock	21-311		
Skip While Erasing	10-103		
Remove Logical Interlock	07		
Terminate Function	23		

19.5.5. EXEC I/O ERROR MESSAGES (Continued)

Function Codes Displayed by I/O Error Messages: (Continued)

Magnetic Drum
- 02 - Write Continuously
- 23/33 - Terminate
- 40/50 - Bootstrap
- 42 - Read Continuously
- 52 - Read Block & Term EOB
- 45 - SRH;FIND; Term SRH -
- 46 - SRH Read;FIND, Cont Read
- 55 - BLK SRH Read;FIND,CONT READ EOB

Cards
- 51 - Input: XFER 1
- 52 - Input: XFER 1, Trip Fill
- 53 - Input: Trip 1
- 70 - Input: Select Stacker 1
- 71 - Input: Select Stacker 2
- 72 - Input: Enable Translate
- 73 - Input: Read by Column
- 74 - Input: Read by Row

High Speed Printer
- 12 - Print
- 33 - Terminate

Paper Tape
- 04 - Punch On
- 05 - Reader On
- 06 - CPR/Error Cl
- 07 - Punch Off
- 08 - Reader Off
- 00 - Read Forward
- 01 - Read Backward
- 02 - Fault
- 03 - Master Clear

19.5.6. TERMINATION CODES

Upon occurrence of the conditions listed below, the EXEC performs an error termination of the job program and provides a typeout to identify the packet of instruction address causing the error. The error codes listed below are included in the typeout.

The format of the EXEC termination as shown on termination log is as follows:

EXEC///time EXEC TER *jrid* (the last re-entry point for the program)
ERROR CODE

1. Illegal jumps and interrupt lockout

Octal code	Meaning
00	Program has executed a jump or return jump to address 0 or 1
75	Program has jumped to an interrupt location
77	Manual termination of program in instruction loop or in loop with interrupts disabled

2. Internal error interrupt

Octal code	Meaning
01	Illegal operation
02	Trace Mode
03	Memory lockout
05	Characteristic underflow
06	Characteristic overflow
07	Divide overflow
51	Illegal operation and illegal recovery address in $ERROR table
52	Trace mode and illegal recovery address in $ERROR table
55	Characteristic underflow and illegal recovery address in $ERROR table
56	Characteristic overflow and illegal recovery address in $ERROR table
57	Divide overflow and illegal recovery address in $ERROR table
61	Illegal trace mode set in area other than current program or EXEC entrance
65	EXEC termination because *jrid* exceeded run time

For nonrecoverable error interrupt conditions, the appropriate code and P register setting are stored in the B0 film image for the terminated program.

19.5.6. TERMINATION CODES (Continued)

COMMUNICATIONS

Octal Code	Meaning
11	Exceeded number of packets allowed for chain
12	Packet or buffer address outside of program limits
14	Illegal unit or channel select
15	Illegal function code

For communications error conditions, the appropriate code and packet address are stored in the B0 film image for the terminated program.

PROGRAM RELEASE

Octal Code	Meaning
16	Illegal release (test instruction error)
17	No outstanding requests at time of release
76	Job program has tested an illegal packet before releasing control to $WAIT1

For program release error conditions, the return address from the B1 register and appropriate code are stored in the B0 film image for the terminated program.

I/O

The I/O error termination codes for those persons using EXEC I/O can be found in the Executive I manual, Appendix F.

ILLEGAL ADDRESS

Octal Code	Meaning
74	One or more words of a transfer by the Q0 register lies outside the core area assigned to the job program.

19.5.7. SORT ERROR CODES

ERROR NUMBER	ERROR DESCRIPTION
00	Overlay segment not read from drum properly.
01	Input label checking requested but label not supplied in schedule for checking.
02	Total of individual key sizes specified on PM1 does not equal the total key size specified on PM0. See also error 12, because if segment 6 is being executed, error 12 is typed out as error 02.
03	Some error was determined in the parameter cards.
04	Parameter for type of input media not a legal code.
05	UNISERVO IIA tape unit specified as input but no IIA tape units assigned by EXEC.
06	UNISERVO IIIA tape unit specified as input but no IIIA tape units assigned by EXEC.
07	UNISERVO IIIC tape unit specified as input but no IIIC tape units assigned by EXEC.
08	No input media was specified by this parameter decode routine.
09	Number of UNISERVO IIA tape units and UNISERVO IIIC tape units specified by PM0 does not equal amount assigned by EXEC.
10	Number of UNISERVO IIIA tape units specified by PM0 does not equal amount assigned by EXEC.
11	Non-numeric character in field being converted from Fieldata to binary, parameter decode.
12	External format specified as input but item size not correct. Must be 14 or 22.
13	Item size too large to be sorted with this amount of core.
14	Did not complete change tape typeout properly; an answer other than Y or YES was returned to the change tape typeouts (input tape; final output).

19.5.7. SORT ERROR CODES (Continued)

ERROR NUMBER	ERROR DESCRIPTION
15	Not enough core is available to handle this type of intermediate storage, because buffer size is below lower limit (120 words).
16	Tapes used as intermediate storage, but not enough core has been assigned to handle all required buffers.
17	Number of tapes specified for merge is greater than the amount of UNISERVO tape units available.
18	Sequence error during Cascade Merge; previous winner was locked and still chosen as winner.
19	Sequence error during Cascade Merge; core location was locked and still chosen as winner.
20	Level count error detected. Maximum number of levels in merge tournament has been exceeded.
21	Sequence error on final output.
22	Merge cannot handle all strings on the drum.
23	Error detected in preparations for handling strings in a partial collation.
24	Error in partial collation.
25	Error detected in Cascade Merge. Attempt to merge more than the available number of tapes.
26	Error detected in attempting to merge more strings than available during partial collation.
27	Last pass of merge has detected that there are no strings to be merged.
28	Error on converting Fieldata to binary, entire card or field missing, parameter decode routine.
29	EOF exit was taken when the first item was requested from a file.
30	Read error, if tape mark tape bad.

19.5.7. SORT ERROR CODES (Continued)

ERROR NUMBER	ERROR DESCRIPTION
31	Input item size does not correspond to item size on parameter card.
32	Error typeout not completed properly.
33	The sign of the numeric key is not a Fieldata +, -, or space.
34	Illegal key tape
35	Illegal function
36	Trace mode
37	Memory Lockout
38	Not used
39	Character underflow
40	Character overflow
41	Divide overflow
42	Not used
43	Too many instructions have been generated for Pack and Unpack.
44	Total key from PM0 does not equal total of keys on PM1.
45	IBM key format not available.
46	File name of PM3 and PM6 cards (external format) and tape do not match.
47	Reel number on PM3 and PM6 cards and tape do not match.
48	Non-numeric reel number on PM3.
49	PM6 card present when none allowed, or missing when IIA output.
50	Spaces in reel number when reel number is to be checked.
51	Non-numeric reel number on output.

EXEC $ERROR Table

19.5.7. SORT ERROR CODES (Continued)

ERROR NUMBER	ERROR DESCRIPTION
52	First internal error interrupt (unassigned).
53	No items entered into Sort.
54	Illegal parameter (2) on indication of output for external and special external format.
55	Drum only specified, but back up tapes are needed.
56	Call was made on OCFP or OCLP in the SORTN version which does not allow Own Coding.
57	The output media specified on PM0 were not assigned to the program.
63	A key specified on the PM1 card extends beyond the length of the item.
64	Duplicate entry on PM2 card.
65	Drum is specified for intermediate storage but no drum table is defined.
66	Drum is specified for final output but no drum table is defined.
67	Tape units are needed as intermediate storage and none are assigned, or not enough tapes.
68	Key fields on PM1 card overlap.
69	No input tape assigned.
76	Error on input positioning of multi-file input.
77	End of tape (IIA internal format)
90 or 128	Non-alpha/numeric on PM0 or PM1. (00-77)
91	Merge did not allow enough core for 2 buffers per input and 2 buffers per output file.
92	PM2 is used but total bits are not divisable by six.

19.5.8. ELF ERROR CODES

CODES	DESCRIPTION
01	Drum read error
02	Checksum Error
03	Machine or peripheral equipment error
04	Drum write error
05	Tape read error
06	Tape write error
10*	Drum STABP or ENTRYP not long enough (See next page)
12	Element is not RB
13	Illegal INFO statement
14	Drum logical channel not available
15	FASTRAND not available
16	Negative sign in code
20	Name on control card contains a space
21	Element card format error
22	Element does not contain a preamble
23	Too many characters in a name or control card
24	Control card format error
25	PREAMB core table overflow
26	No valid preamble cards (col 1 neg - element skipped)
27*	CRBTLS drum table overflow
30*	DRMPOS drum table overflow
31*	DELETE core table overflow
32*	STAB core table overflow
33*	ENTRY core table overflow
34	SUB core table overflow
35	ENT core table overflow
36	RB entry point not in ENTRY table
37*	CRSRF table overflow
40*	PROGF table overflow
41	Translation error
42	Entry not in STAB table
43	GSEO table overflow
44	SYS table overflow
45	Element name not an entry point
46*	PREAMB table overflow (printing)
47	More than one Map card
50	Reference not defined
51	Right half modification not allowed
52	Left half modification not allowed
53	12 bit modification not allowed
54	I/O table overflow
55	Drum table overflow
56	Core table overflow
57	JS table overflow

19.5.8. ELF ERROR CODES (Continued)

CODES	DESCRIPTION
60	No ROC input tape assigned
61	Alternate ROC input tape assigned and main ROC input not assigned
62	ROC label block checksum error
63	ROC first block not label block
66	ROC block not 256 words
67*	CRSRF table overflow
70*	Preamble table overflow (Library Directory)
71	Too many entry points (more than 50 entry points in element)
72*	PLTAB table overflow
73	Illegal equipment type
74	IB, DB, or data table over 65K

Errors marked in the above list are recoverable through rescheduling. An increase in table length (either core or drum) is required. This is done by a use of a TAL card in the Job Request for ELF. The following table lists those errors which are of the recoverable nature.

CODES	TABLE TYPE	LENGTH TAG
10_{xy}	DRUM	STABP or ENTRYP
27	DRUM	CRBTL
30	CORE	DRUMPSL
31_{yx}	CORE	DELETL
32	CORE	STABL
33_{xx}	CORE	ENTRYL
37 or 67	CORE	CRSRFL
40	DRUM	PROGFL
46 or 70	CORE	PRE
72	CORE	PLTABL

x If the length of ENTRYL (core) is increased by an increment at load time, it is also necessary to lengthen ENTRYP (drum) by the same increment.

y If the length of STABL (core) is increased by an increment at load time, it is also necessary to lengthen STABP (drum) by the same increment.

19.5.9. FILE CONTROL ERRORS

All file control errors will be displayed on the console typewriter in the format: File-name FC ERR XXX

where XXX is the value associated with the particular error. A complete list of file control errors is in the LION - General Reference Manual, UP-2581, Rev. 2.

After the typeout has been displayed, LION will either terminate the run or return control to the object program. This action is determined by the open file statement.

The following LION file control errors have no recovery procedures available to the FORTRAN program. These errors when detected will result in termination via EXEC $ERR.

F. C. ERR CODE	DESCRIPTION
5	An attempt to perform a CALL ITMOUT statement has been made, the file closed or demanding a CALL OENREL operation.
6	An attempt to perform a CALL ITMIN statement with the file not yet opened, or when a CALL IENFIL or IENREL is the only legitimate operation.
7	An attempt has been made to execute the End-of-Reel procedure on a closed input file.
8	An attempt has been made to execute the End-of-Reel procedure on a closed output file.
118	The file data area being requested is occupied by a previously opened file.
119	The number of opened files exceeds 20.
120	When opening an output file, one of the required arguments (A,B,C or E) was not present. For opening an input file (A,B,C, or D) was not present. For ITMIN (A,B, or C) is not present. For ITMOUT (A or B) is not present. For any other operations, Argument A is not present.
121	The peripheral equipment definition is not a legal value. (Peripheral definitations are defined in NTAB$)
122	In an item read or write, a close reel or file, or an ITMRTN statement, the file data area requested is not defined.

19.6. CHANNEL ASSIGNMENTS

19.6.1. CENSUS 1107 CHANNEL ASSIGNMENTS -- 1107A and 1107B

The following are the I/O channel assignments as of this date for the Census 1107:

Site A = 07 B = 12

Peripheral Unit	Channel Assignment	No. of Units	
FH-880 Drum	0	2	
Uniservo IV C Read/Write	1	2	
Uniservo II A Read/Write	2	2	
Uniservo II A Read/Write	3	2	
Fastrand Mass Storage	5	3	
Uniservo III A Read/Write	6	7	] - same units
Uniservo III A Read/Only	7	7	]
Uniservo III A Read/Write	8	7	] -same units
Uniservo III A Read/Only	9	7	]
Paper tape Subsystem	11 - 0	1	reader
	11 - 8	1	punch
	11 - 9	1	labeler
High Speed Printer	12	1	
Card Reader	13	1	
Console	15	1	

19.6.2. CENSUS 1108 CHANNEL ASSIGNMENTS -- 1108C

The following are the I/O channel assignments as of this date for the Census 1108C:

Site 43

Peripheral Unit	Channel Assignment	No. of Unit	
FH-432 Drum	0	2	
FH-1782	0	1	
Uniservo VIII C Read/Write	1	2	
Uniservo II A Read/Write	2	2	
Uniservo II A Read/Write	3	2	
FH-880 Drum	4	2	
Fastrand Mass Storage	5	3	
Uniservo III A Read/Write	6	7	same units
Uniservo III A Read/Write	7	7	
Uniservo III A Read/Write	8	7	same units
Uniservo III A Read/Only	9	7	
Paper tape Subsystem	11 - 0	1 reader	
	11 - 8	1 punch	
	11 - 9	1 labeler	
High Speed Printer	12	1	
Card Reader	13	1	
Console	15	1	

19.6.3. CENSUS 1108 CHANNEL ASSIGNMENTS -- 1108D

The following are the I/O channel assignments as of this date for the Census 1108D:

Site 37

Peripheral Unit	Channel Assignment	No. of Units
FH-432 Drum	0	2
FH-1782 Drum	0	1
Uniservo VIII C Read/Write	1	2
Uniservo II A Read/Write	2	2
Uniservo II A Read/Write	3	2
Fastrand Mass Storage	5	3
Uniservo III A Read/Write	6	0] - same units ☆
Uniservo III A Read/Only	7	0]
Uniservo III A Read/Write	8	14] - same units
Uniservo III A Read/Only	9	14]
Paper tape Subsystem	11 - 0	1 reader
	11 - 8	1 punch
	11 - 9	1 labeler
High Speed Printer	12	1
Card Reader	13	1
Console	15	1

☆ Channel (6) must be downed after booting on this machine.

19.7. UNIVAC 1107 STANDARD CODE TRANSLATION

UNIVAC 1107
STANDARD CODE TRANSLATION

FIELDATA

OCTAL CODE	1107 HSP CHARACTER	CODING SYMBOL	KEY	PUNCH	NAME	MONITOR KEY-BOARD	MONITOR PRINTER	OCTAL CODE	XS3 CHARACTER	NAME
00	@	@		7-8	at	≈	(IGN)	00	÷	ignore
01	[	[		12-5-8	left bracket	UC	↴	01	△	space
02	]	]		11-5-8	right bracket	LC	⁄	02	–	minus
03	#	#		12-7-8	number sign	LF	(L.F.)	03	0	
04	△	△		11-7-8	delta	Return	(C.R.)	04	1	
05	(blank)	blank or ß	space			Space Bar	(Space)	05	2	
06	A	A	A	12-1		A	A	06	3	
07	B	B	B	12-2		B	B	07	4	
10	C	C	C	12-3		C	C	10	5	
11	D	D	D	12-4		D	D	11	6	
12	E	E	E	12-5		E	E	12	7	
13	F	F	F	12-6		F	F	13	8	
14	G	G	G	12-7		G	G	14	9	
15	H	H	H	12-8		H	H	15	'	apostrophe
16	I	I	I	12-9		I	I	16	&	ampersand
17	J	J	J	11-1		J	J	17	(	left parenthesis
20	K	K	K	11-2		K	K	20	+	multiline
21	L	L	L	11-3		L	L	21	,	comma
22	M	M	M	11-4		M	M	22	.	period
23	N	N	N	11-5		N	N	23	;	semicolon
24	O	O	O	11-6		O	O	24	A	
25	P	P	P	11-7		P	P	25	B	
26	Q	Q	Q	11-8		Q	Q	26	C	
27	R	R	R	11-9		R	R	27	D	
30	S	S	S	0-2		S	S	30	E	
31	T	T	T	0-3		T	T	31	F	
32	U	U	U	0-4		U	U	32	G	
33	V	V	V	0-5		V	V	33	H	
34	W	W	W	0-6		W	W	34	I	
35	X	X	X	0-7		X	X	35	#	number sign
36	Y	Y	Y	0-8		Y	Y	36	¢	cent
37	Z	Z	Z	0-9		Z	Z	37	@	FF1

19.7. UNIVAC 1107 STANDARD CODE TRANSLATION (CONTINUED)

40	)	)	⌑	12-4-8	right parenthesis	)	)	40	+	tab
41	—	—	—	11	minus	—	—	41	"	quote
42	+	+	&	12	plus	+	+	42	I	FF2
43	<	<		12-6-8	less than	<	<	43	)	right parenthesis
44	=	=	#	3-8	equals	=	=	44	J	
45	>	>		6-8	greater than	>	>	45	K	
46	&	&		2-8	ampersand	—	—	46	L	
47	$	$	$	11-3-8	dollar sign	$	$	47	M	
50	*	*	*	11-4-8	asterisk	*	*	50	N	
51	(	(	%	0-4-8	left parenthesis	(	(	51	O	
52	%	%		0-5-8	percent	"	"	52	P	
53	:	:		5-8	colon	:	:	53	Q	
54	?	?		12-0	question mark	?	?	54	R	
55	!	!		11-0	exclamation point	!	!	55	$	dollar sign
56	,	,	,	0-3-8	comma	,	,	56	*	asterisk
57	\	\		0-6-8	left slash	(S)	(S)	57	?	FF3
60	0	0	0	0		0	ø	60	Σ	printer stop
61	1	1	1	1		1	1	61	B	break- point
62	2	2	2	2		2	2	62	:	colon
63	3	3	3	3		3	3	63	+	plus
64	4	4	4	4		4	4	64	/	slash
65	5	5	5	5		5	5	65	S	
66	6	6	6	6		6	6	66	T	
67	7	7	7	7		7	7	67	U	
70	8	8	8	8		8	8	70	V	
71	9	9	9	9		9	9	71	W	
72	'	"	@	4-8	apostrophe	'	'	72	X	
73	;	;		11-6-8	semi-colon	;	;	73	Y	
74	/	/		0-1	right slash	/	/	74	Z	
75	.	.	.	12-3-8	period	.	.	75	%	percent
76	⌑	⌑		0-7-8	lozenge	SPEC	□	76	=	FF5
77	(stop)	≠		0-2-8		↓	↓	77	Not Used	

(IGN) = Ignored *(F.F.) = Fast Feed* *0-3-8 = Comma*
(C.R.) = Carriage Return *(L.F.) = Line Feed* *4-8 = Apostrophe*

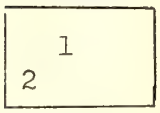
19.8. BCD, EXCESS-THREE CODES

Numeric Bits

	00	01	10	11
0000	i 5 00	r E 20	t N 40	Σ V 60
0001	△ 6 01	, 21	" 0 41	β W 61
0010	- 02	.. 22	/ P 42	: 62
0011	0 03	; 23	) 43	+ 63
0100	1 04	A 24	J 44	/ 64
0101	2 05	B 25	K 45	S 65
0110	3 06	C 26	L 46	T 66
0111	4 07	D 27	M 47	U 67
1000	5 10	E 30	N 50	V 70
1001	6 11	F 31	O 51	W 71
1010	7 12	G 32	P 52	X 72
1011	8 13	H 33	Q 53	Y 73
1100	9 14	I 34	R 54	Z 74
1101	' 15	# 35	$ 55	% 75
1110	& 16	¢ C 36	* 56	= T 76
1111	(17	@ D 37	? M 57	NOT USED

Zone Bits

FIELDS



HIGH SPEED PRINTER

PRINTING

Normal: Prints Field 1 except for non-printing characters (see Computer Digit).
Computer Digit: prints Field 1 for printing characters; Field 2 for non-printing characters.

ACTION

@ Fast Feed Symbol 1
/ Fast Feed Symbol 2
? Fast Feed Symbol 3
= Fast Feed Symbol 4
r Multi-line Symbol
i Printer Ignore
△ Printer Space
β Breakpoint Stop
Σ Printer Stop

NOTE: The numbers in the lower right hand corner of each box are the octal equivalen of the XS-3 character.

19.9. TABLE OF POWERS OF 2

TABLE OF POWERS OF 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

19.10. CAPACITY OF TAPES WRITTEN WITH FOLION

IIA TAPES

The block limit is 5000 data blocks.

A block contains 120 words.

The total word capacity is 600,000.

IIIA TAPES

There is no specific block limit. Data is written until end-of-tape or a programmed end-of-file sentinel.

Recording is 200 words per inch with .75 inch interblock spacing. An 1800' reel with block size of 605 may contain approximately 5000 blocks or 3,000,000 words of data.

19.11. STANDARDS FOR OPERATION AND USE OF UNIVAC 1107 SYSTEM

Analysis of facility requirements of computer programs utilizing the Univac 1107 system indicates that certain additional facility allowances can be made for individual programs without significant adverse effect on the system's ability to undertake multi-program concurrent operations.

Facility allowances

The initial standards for 1107 computer use, dated June 4, 1963, are modified, therefore to provide the following maximum allowances for individual programs using the 1107 under the circumstances indicated.

Program Type	Core Storage	Drum Storage	IIIA	IIA 1/	VIIIC IVC 2/
1. Production programs other than sorts	26 blocks (53,248 words)	1,000,000 words	6	1	1
2. Sorts with no additional option coding outputs	26 blocks (53,248 words)	1,000,000 words	6	1	1
3. Sorts with additional option coding outputs	26 blocks (53,248 words)	1,000,000 words	6	1	1

It should be noted that sorts which depend upon the maximum drum storage allowance of 1,000,000 words may, on occasion, have to be run with reduced drum facility if one of the FH-880 drums is out of service. If the volume being sorted at the time requires the maximum drum storage allowance to eliminate extra merge passes, the running time will be increased during the operation with the reduced drum facility.

Run size

Normally, a maximum of 4,500,000 words can be sorted efficiently with the maximum facilities indicated, including five intermediate Uniservos. If the data for a single sort exceed that number, the extra merge passes may result in running time in excess of that for two sorts and one merge of the same amount of data.

There have been some requests for a recommended maximum run time. One hour of central computer time is the recommended maximum for continuous operation between recovery points. This is not a standard in the same sense as those for maximum facility utilization but it is a practical limit beyond which the user must anticipate a higher rate of non-productive time.

1/ IIA tape units will be phased out of the computer systems once replacement high speed printers are installed and FOSDIC-70 tape units are replaced by compatible tape units. Therefore direct use of Uniservo IIA in a production program should be avoided to the extent possible. An exception will be granted for programs using FOSDIC-70 output until hardware revisions are made to the FOSDICs.

2/ Use is restricted to one half the available units (2 at present).

19.11. STANDARDS FOR OPERATION AND USE OF UNIVAC 1107 SYSTEM (Continued)

On-line printing facilities

In order to reduce the volume of "RUTT" translations now required to permit off-line printing of computer output, provision will be made for on-line printing of limited volumes of test and/or production output. Although it will not be possible for a programer to call for assignment of the on-line printer to an individual program, he may cause small quantities of output to be prepared for separately scheduled on-line printing through the CENSER service routine. Provision is being made to print up to a maximum of 5,000 lines in this manner, the maximum to be considered as applying to the total of all the output files of a single run for which this technique for printing is to be used.

To use the facility the programer may write the data in one of two ways:

1. In LION external format, Fieldata code, on a Uniservo IIIA; option TP.

2. In Census format LION, BCD XS-3, on a Uniservo IIA.

The operating instructions for a program must specify on-line printing and must state specifically which of the two formats is to be printed by CENSER. Operations personnel will exercise the option to print some of this work off-line in the event of a backlog of on-line printing. To reduce the possibility of having such a backlog, the facility should be used primarily for limited amounts of test or production printing.

Reels normally printed off-line will be printed on-line, at the discretion of scheduling personnel, provided the following conditions are met:

1. A Standard board (1-5), standard loop (1-4), and Forms 143-1, 2, 3, or 4 (or comparable form) are requested.
2. The on-line HSP or IIA tape unit are not needed for other processing, including regular on-line printing.
3. The backlog for off-line printing is more than 100 reels.

19.12. MOST USED BOARDS AND LOOPS FOR OFFLINE HSP

BOARD #1 Ten 12-digit words per line - Space between words - No zero suppression.

BOARD #2 Ten 12-digit words per line - Space between words - Zero suppression on each word.

BOARD #3 Ten 12-digit, or twenty 6-digit, words per line - No space between words - No zero suppression.

BOARD #4 Ten 12-digit words per line - No space between words - Zero suppression on each word.

BOARD #5 Twenty 6-digit words per line - No space between words - Zero suppression on each word.

PAPER LOOPS

LOOP #1 Space after six double-spaced print lines - 24 print lines per 11-inch page.

#2 Space over fold - 31 double-spaced print lines per 33-line page.

#3 Space over fold - 30 double-spaced print lines per 33-line page.

#4 Space to new fold for fastfeeds 1, 2, 3, or 4

Note: There are more than 11 boards and numerous paper loops available upon request. These are the most frequently used.

When printing binary data (not XS3 characters), ask for Board 3 and binary loop. It is also possible to ask for computer digit printing where all non-printable characters will be printed on a second line with substitutions made for various bit combinations.

Custom boards and loops may be requested with discretion. Forms BC-1596 and BC-1170 should be completed to conform to specifications. Consult the manual "The High Speed Printer of the UNIVAC System RRU16.1" and contact the shift supervisor in Computer Operations Branch of Processing Division.

19.13. HSP NON-PRINTING CHARACTERS

The following characters do not print, however a representative character is printed when Computer Digit is specified, for each of the respective characters.

ACTION	SYMBOL	COMPUTER DIGIT	BINARY	OCTAL
ignore	ɨ	5	000000	00
multiline	ɾ	E	010000	20
tab	t	N	100000	40
printer's stop	ɬ	V	110000	60
space	∆	6	000001	01
breakpoint	β	W	110001	61
fastfeed 1	@	D	011111	37
fastfeed 2	|	P	100010	42
fastfeed 3	?	M	101111	57
fastfeed 4	=	T	111110	76

19.14. AMOUNT OF SPACE PER PAGE

Printer	No. of Characters per Line	No. of Lines per Page
1107 On-Line	128	54
UNIVAC Off-Line	120	60
IBM 1401	132	54

Note: When printing off-line, a "fastfeed" before the first output and after the last output will prevent the sentinels on the IIA tape from appearing on the same pages as the output. The octal fastfeed code should be inserted into the first character of a line in order to take effect.

19.15. DATA PROCESSING EQUIPMENT AT THE BUREAU OF THE CENSUS (Subject to change)

2 UNIVAC 1108 COMPUTERS

131 K (Sys. D) 65 K (Sys. C) Word Core Memory

2 FH432 Drums (524,288 words)

1 FH1782 Drum (2,097,152 words)

3 FASTRAND II Units (66,060,288 words shared)

14 UNISERVO IIIA Tape Units (9-track)

4 UNISERVO IIA Tape Units (8-track)

2 UNISERVO VIII C Tape Units (IBM Comp. 7-track)

1 Punch-Card Reader

1 Paper-Tape Reader

1 Paper-Tape Punch

2 High-Speed Printers (755 Sys. C, 758 Sys. D)

1 CONSOLE PRINTER

1 REMOTE TELEPRINTER

19.15. DATA PROCESSING EQUIPMENT AT THE BUREAU OF THE CENSUS (Subject to change) (Continued)

2 UNIVAC 1107 COMPUTERS

128-Word Thin Film Memory

65 K Word Core Memory

2 FH 880 Drums (1,572,864 words)

3 FASTRAND II Units (66,060,288 words shared)

14 UNISERVO IIIA Tape Units (9-track)

4 UNISERVO IIA Tape Units (8-track)

2 UNISERVO IV C TAPE UNITS (IBM Comp. 7-track)

1 Card Reader

1 Paper-Tape Reader

1 Paper-Tape Punch

1 High-Speed Printer

1 CONSOLE PRINTER

1 REMOTE TELEPRINTER

1 LABEL PRINTER

19.15. DATA PROCESSING EQUIPMENT AT THE BUREAU OF THE CENSUS (Subject to change) (Continued)

4 UNIVAC HIGH SPEED PRINTERS

Off-Line IIA Input

2 MOHAWK DATA SCIENCES (MDS) 7160 OFF LINE PRINTER SYSTEMS

1000-1250 lines per minute

132/160 characters per line

556/800 bits per inch (BPI) densities

1024 character core memory

1 MDS 2207 Magnetic Tape Unit per System

1 STROMBERG-CARLSON SC-4411 (Computer Document Recorder)

Converts information on Magnetic IV Tape to microfilm. Translates up to 4 frames (76 lines of 132 characters each) per second.

1 DATAMEC D 3029

41,700 CPS transfer rate

556 BPI density

2 KALVAR MICROFILM COPIERS

Produce copies of 16 mm microfilm on to Kalvar film (input is produced on SC-4411)

Copies at a rate of 60 to 180 feet per minute

1 VISCOMAT PROCESSOR

Develops film from SC-4411 at rate of 36 feet per second

19.15. DATA PROCESSING EQUIPMENT AT THE BUREAU OF THE CENSUS (Subject to change) (Continued)

1 IBM 360 MODEL 40 COMPUTER

65 K Bytes Core Memory

3 Disk Storage Units (2,250,000 each)

5 2401 Uniservo's (2-7 track, 3-9 track)

1 1403 Printer (1100 line/min)

1 2540 Punch-Card Reader (Read 1000/min., Punch 300/min.)

1 Central Processing Unit

1 Console (1052)

1 Data Transmission System 1013-7702 between Jeffersonville and Suitland

1 IBM 1401 TAPE COMPUTER

1 2K Character Core Memory
1 1402 Card-Read Punch
1 1403 Printer
2 7330 Tape Units (7-track)
1 1922 Tape Adapter Unit
2 Uniservo Tape Units (8-track)
1 1407 Console Inquiry Station
1 1627 II Plotter

19.15. DATA PROCESSING EQUIPMENT AT THE BUREAU OF THE CENSUS (Subject to change) (Continued)

6 FOSDIC 70's (Film Optical Sensing Device for Input to Computers)

24 Bit Word Size

8K Word Core Memory

1 A Register

1 Fosdic Scan Unit

2 Mark III Uniservo's (UNIVAC IIA Comp. 8-track)

1 Card Reader

1 Flex-o-writer

MISCELLANEOUS

ITN REMOTE TERMINALS - CONVERSATIONAL TERMINALS

DIGIDATA - Creates a magnetic tape from paper tapes prepared by Teletype

CalComp PLOTTER - Draws lines according to plotted positions for use with IBM 1401

DCT-2000 - UNIVAC remote batch (cards and printer under EXEC 8)

19.16. CHARACTER SET FOR INFORMATION INTERCHANGE

When preparing data on magnetic tape for sale, or for another form of interchange, the character set should be restricted to those characters which are generally accepted and which are represented by a unique code in any particular coded decimal language. Analysis of the IBM binary coded decimal tape code recognized by eight computer manufacturers on approximately fifteen different computer types reveals a consensus on the ten decimal digits, the twenty-six alphabetic symbols, six special characters and the blank or space. Also, each of these systems recognize the Tape Mark.

The attached character conversion chart, Appendix I, contains the forty-three character graphics and their coded representation in four languages. The three six-bit languages are shown by their octal representation; the eight-bit language is shown in hexadecimal. Please note that the binary coded decimal, excess three (BCD XS-3), code for space (Δ), octal 01, is not shown. There is no distinction between blank or space in the other languages. Therefore, when preparing potential interchange data in BCD XS-3 originally, use only the ignore (Ɨ), octal 00.

Conversion between BCD XS-3 and binary coded decimal (BCD) is performed by hardware on the UNIVAC 1107 or 1108 and on the IBM 1401 on the premises. Conversion between BCD and extended binary coded decimal interchange code (EBCDIC) is performed by hardware on the IBM 360, Model 40. Conversion between BCD XS-3 and Fieldata is performed by subroutine ZC01. Conversion between Fieldata and BCD must be programmed.

The Tape Mark, which exists on IBM compatible seven or nine channel tapes, is produced by hardware in response to an instruction. It is not obtained by conversion from BCD XS-3 or Fieldata.

The six special characters which may be used in data intended for interchange are indicated below. They appear also in Appendix I.

character graphic	octal BCD XS-3	octal BCD	octal Fieldata	hexa-decimal
- (minus)	02	40	41	60
, (comma)	21	33	56	6B
. (period)	22	73	75	4B
$ (dollar sign)	55	53	47	5B
✯ (asterisk)	56	54	50	5C
/ (slash or virgule)	64	21	74	61

19.16. CHARACTER SET FOR INFORMATION INTERCHANGE (Continued)

APPENDIX I

Conversion Code for 43 Characters in Four Coded Decimal Languages

Character graphic	Octal 1/ BCD XS3 5/	Octal 2/ BCD 5/	Octal 3/ Fieldata 5/	Hexadecimal 4/ EBCDIC 6/
blank (space)	00	20 7/	05	40
- (minus)	02	40	41	60
0	03	12	60	F0
1	04	01	61	F1
2	05	02	62	F2
3	06	03	63	F3
4	07	04	64	F4
5	10	05	65	F5
6	11	06	66	F6
7	12	07	67	F7
8	13	10	70	F8
9	14	11	71	F9
, (comma)	21	33	56	6B
. (period)	22	73	75	4B
A	24	61	06	C1
B	25	62	07	C2
C	26	63	10	C3
D	27	64	11	C4
E	30	65	12	C5
F	31	66	13	C6
G	32	67	14	C7
H	33	70	15	C8
I	34	71	16	C9
J	44	41	17	D1
K	45	42	20	D2
L	46	43	21	D3
M	47	44	22	D4
N	50	45	23	D5
O	51	46	24	D6
P	52	47	25	D7
Q	53	50	26	D8
R	54	51	27	D9
$ (dollar)	55	53	47	5B
✯ (asterisk)	56	54	50	5C

19.16. CHARACTER SET FOR INFORMATION INTERCHANGE (Continued)

Character graphic	Octal 1/ BCD XS3 5/	Octal 2/ BCD 5/	Octal 3/ Fieldata 5/	Hexadecimal 4/ EBCDIC 6/
/ (slash or virgule)	64	21	74	61
S	65	22	30	E2
T	66	23	31	E3
U	67	24	32	E4
V	70	25	33	E5
W	71	26	34	E6
X	72	27	35	E7
Y	73	30	36	E8
Z	74	31	37	E9

1/ This is a 6-bit code known as binary coded decimal excess three, used in earlier model UNIVAC computers and peripheral devices.

2/ This is a 6-bit code known as binary coded decimal, used originally in earlier model IBM computers and peripheral devices.

3/ This is a 6-bit code which originated from an attempt, by the military, to produce a uniform code.

4/ This is an 8-bit code, known as extended binary coded decimal interchange code, which originated with IBM Systems 360.

5/ In the octal representation of each 6-bit code the three high order bits are represented in the most significant digit of the octal code, and the three low order bits are represented in the least significant digit of the octal code. The range of each octal digits is 0-7.

6/ In the hexadecimal representation of the 8-bit code, the four most significant bits are represented in the most significant position of the hexadecimal code, and the four least significant bits are represented in the least significant position of the hexadecimal code. The value range of each hexadecimal character is 0-15, where the values 10, 11 - - - - 15 are defined by the letters A, B - - - F, respectively.

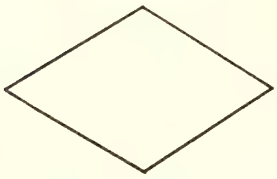
7/ Octal 20 for blank, in BCD, appears only on tape. Internally this code is 00 and the difference between the internal and external representations is handled automatically by the hardware. Where BCD is generated internally in the UNIVAC 1107 or 1108, and written to tape without the hardware translation, the octal 20 must be created in core.

19.17. Flowchart Symbols

Some flowchart symbols are used to indicate the flow and the details of a program. Others are used to indicate input-output or specialized functions. Still others are used to indicate the overall systems design where several programs are needed to process information. The most commonly used symbols are described below.

PROCESSING

A group of program instructions which perform a processing function of the program.

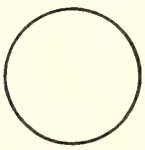


DECISION

The decision function used to document points in the program where a branch to alternate paths is possible based upon variable conditions.

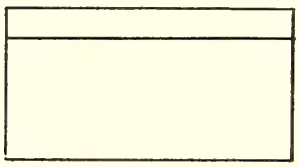
TERMINAL

The beginning, end, or a point of interruption in a program.



CONNECTOR

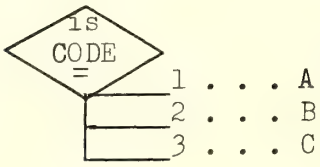
An entry from, or an exit to, another part of the program flowchart. Names consisting of letters or numerals may be assigned.



SYMBOL STRIPING

The striping of a symbol indicates that a more detailed representation of the function can be found elsewhere in the same set of flowcharts.

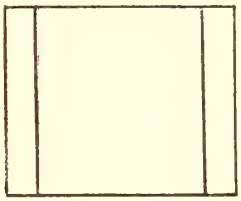
MULTIPLE EXITS



Each condition and each connector must be indentified either individually or by means of a table.

CODE = TO	1	2	3
GO TO	A	B	C

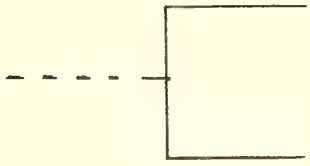
CODE = TO	GO TO
1	A
2	B



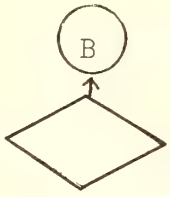
PREDEFINED PROCESS

A group of program steps that are specified elsewhere, e.g., subroutine or logical unit.

ANNOTATION

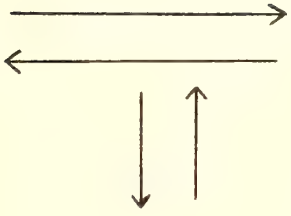


Addition of descriptive comments or explanatory notes as clarification. The broken line may be drawn in any direction. Descriptive information included with the symbols must read from left to right and from top to bottom. For example:



IF OVERDUE CALCULATE SERVICE CHARGE

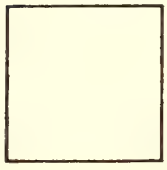
FLOW DIRECTION



Flow direction is shown by lines drawn between symbols. The normal direction flow is from left to right or top to bottom. When this flow is otherwise, arrowheads should be placed in the reverse direction.

INPUT/OUTPUT

Making available information for processing or the recording of processed information.

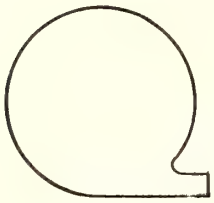


AUXILIARY OPERATION
An offline operation not under control of the central processing unit.

MANUAL OPERATION

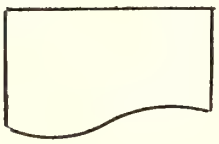
Represents any offline process geared to the speed of the human being

SPECIALIZED INPUT OUTPUT-SYMBOLS

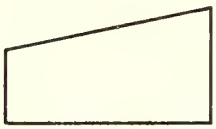


MAGNETIC TAPE

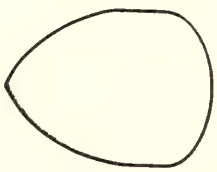
PUNCHED CARD



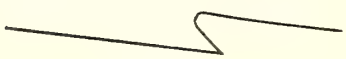
DOCUMENT



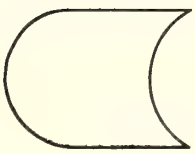
MANUAL INPUT



DISPLAY ONLINE



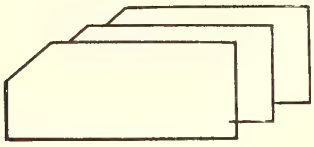
COMMUNICATION LINK



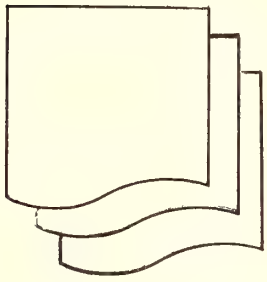
ONLINE STORAGE

OFFLINE STORAGE

MULTIPLE FILES

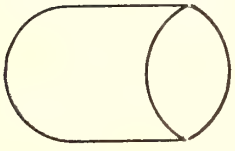


Construct the first symbol and then offset the succeeding symbols toward one direction.

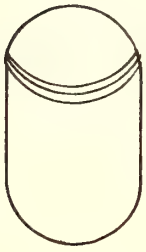


INTERNATIONAL SYMBOLS

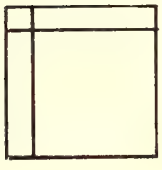
PREPARATION



MAGNETIC DRUM

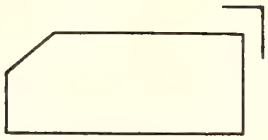


MAGNETIC DISK

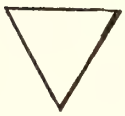


CORE

DECK OF CARDS

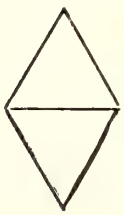


FILE OF CARDS

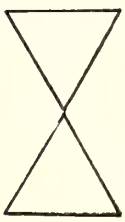


MERGE

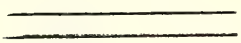
EXTRACT



SORT



COLLATE



PARALLEL MODE

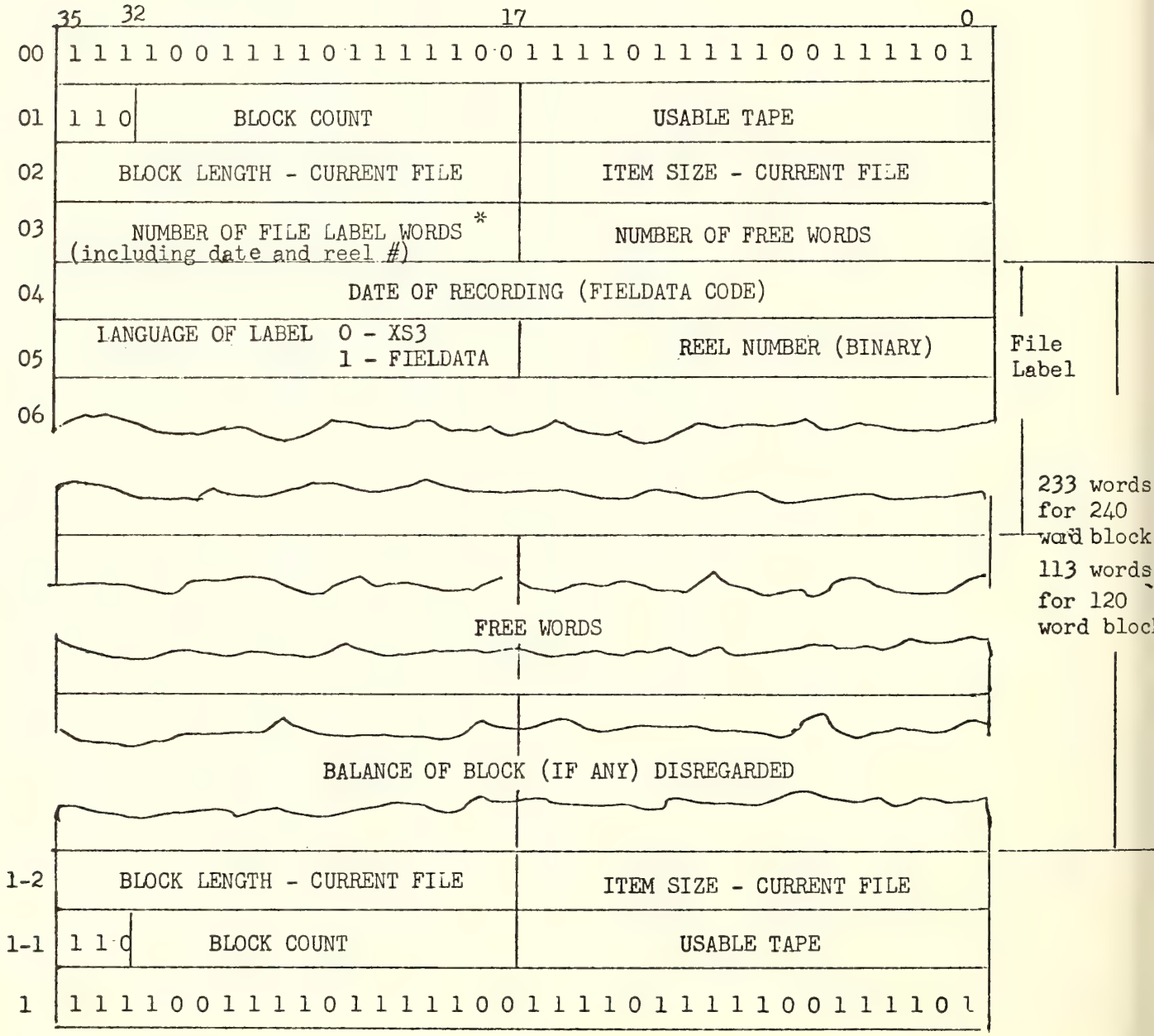
19.18 NAMES TO BE AVOIDED FOR EXTERNAL SUBROUTINES

Sometimes a FORTRAN IV program compiles with no apparent errors and yet an external subroutine prepared by the programmer is not listed in the ELF allocation. This problem may be caused by a conflict of names with system external labels; therefore, the following labels s ould not be used for subroutine names.

A	CEXP	DMIN1	INT	LDSRC	MOVET	SIGN
AØ	CHANGE	DMOD	ITMIN	LDSRL	NFLAG	SIN
A1	CHKDIV	DPADD	ITMIO	LFB	NOCHK	SINCOS
A2	CHSNCO	DSIGN	ITMOUT	LFX	NOTAPE	SINH
A3	CLOCK	DSIN	ITMRTN	LIFTB	NRCK	SINHCO
A4	CLOG	DSINCO	INSERT	LOC	OCSUB	SLITE
A5	CMPLX	DSINH	INSRTH	LOAD	OENFIL	SLITET
ABS	COMPL	DSQRT	INSRTT	LPMØ	OENREL	SNGL
ACOS	CONJG	DTAN	INSRTS	LPM1	OPECLO	SORT
AIMAG	COS	DTANH	ISIGN	LPM2	OPNIN	SQRT
AINT	COSH	DVCHK	IXTRCL	LPM3	OPNOUT	SSWTCH
ALOG	CRIAST	ECSCAN	IXTRCN	LPM4	OR	STORE
ALOG1Ø	CSIN	EOFSET	IXTRCR	LPM5	OVERFL	STRIP
AMAXØ	CSINCO	ERR	JAH	LPM6	PACK	T1
AMAX1	CSINH	EX	JAT	LPM7	PART	T2
AMINØ	CSQRT	EXIT	JNH	LPM8	PNCHU	T3
AMIN1	CTAN	EXP	JNS	LPM9	PRCNG	TAN
AMOD	CTANH	F4EDIT	JNT	LSLC	PRNTU	TANH
AND	D	FB	JLH	LSLL	PUNCH	TSTØ
ASIN	DABS	FDIN	JLS	LSRA	R1	TST1
ATAN	DACOS	FDOUT	JLT	LSRC	R2	TRAN
ATAN2	DASIN	FLD	JRH	LSRL	R3	TYPE
B	DATAN	FLOAT	JRS	LSSU	READA	UNPACK
B1Ø	DATAN2	FLOPRP	JRT	LSUB	READC	UNPK
B11	DBLE	FOLION	LADD	LXB	READT	UTABLE
BF	DCBRT	FX	LAND	LXF	REAL	XB
BOOL	DCOS	H1	LB2F	MAXØ	REWINT	XF
BX	DEXP	H2	LB2X	MAX1	S1	XLOC
CABS	DIM	IABS	LBF	MINØ	S2	XM
CBRT	DINT	IDIM	LBX	MIN1	S3	XOR
CCBRT	DLOG	IENFIL	LDSLC	MOD	S4	XSIN
CCOS	DLOG1Ø	IENREL	LDSLL	MOVEH	S5	XSOUT
CCOSH	DMAX1	IFIX	LDSRA	MOVES	S6	XT3

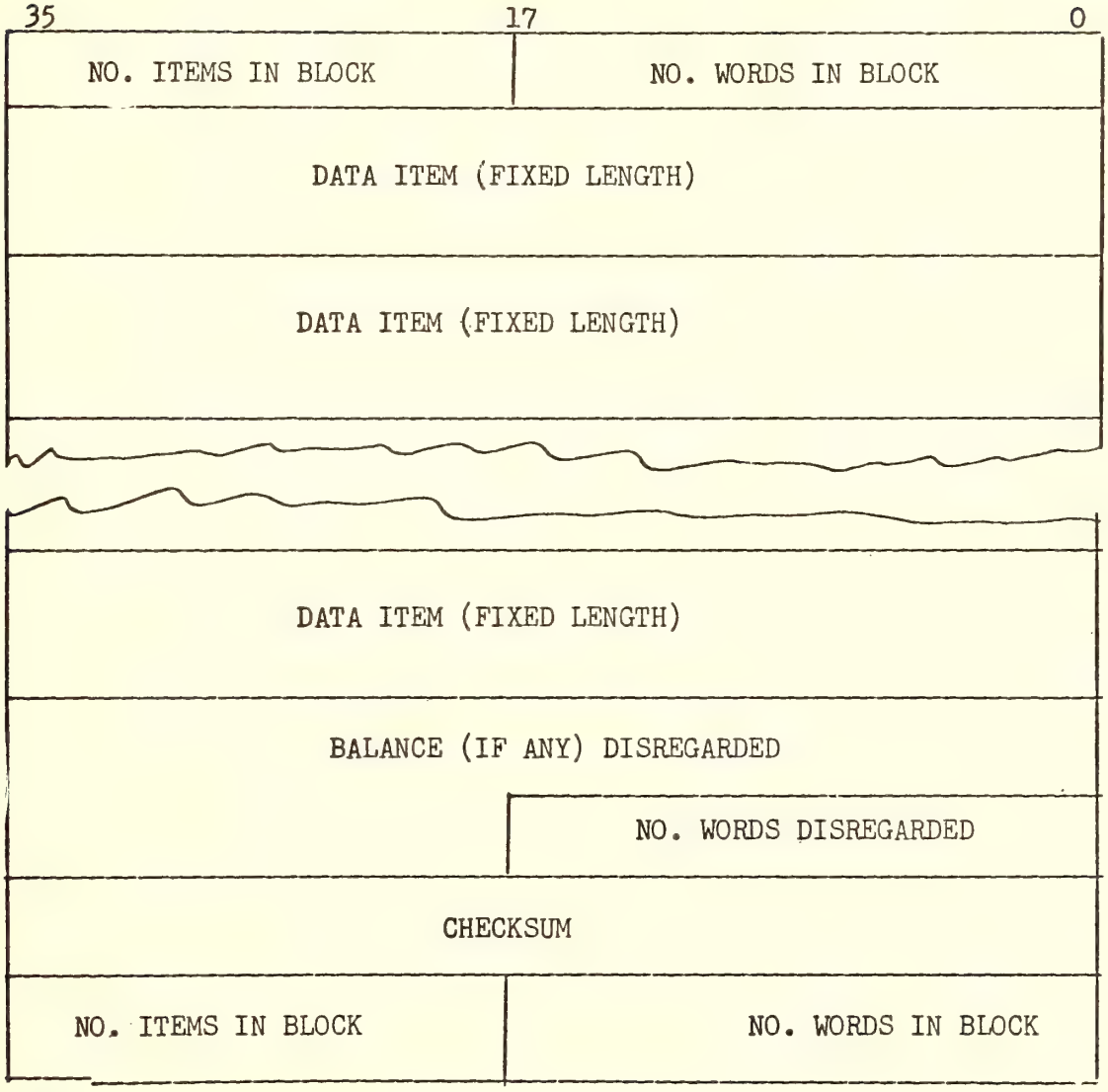
19.19. LION FORMATS

LABEL BLOCK (LION INTERNAL)

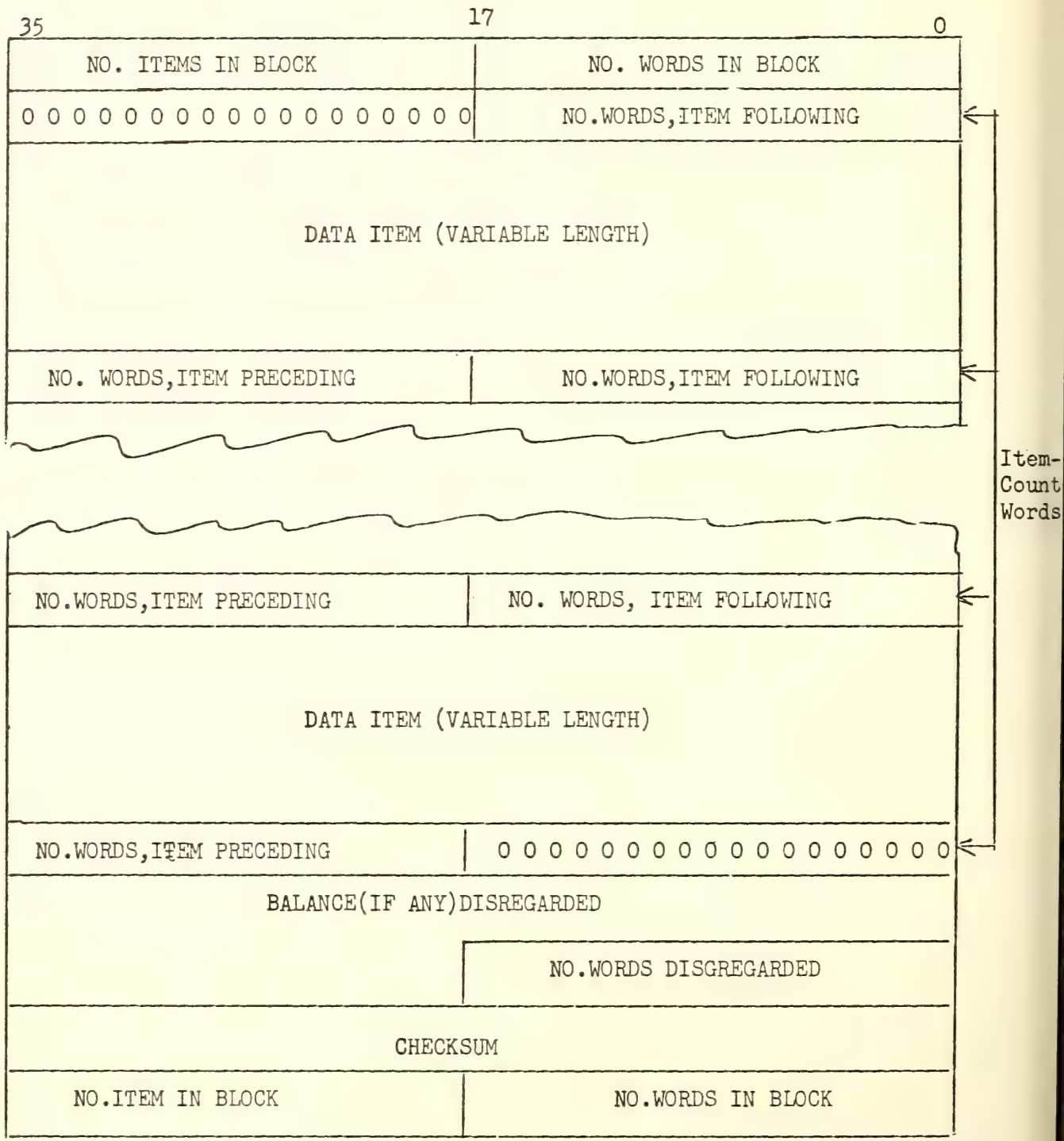

35 32 17 0
00 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1
01 1 1 0 BLOCK COUNT USABLE TAPE
02 BLOCK LENGTH - CURRENT FILE ITEM SIZE - CURRENT FILE
03 NUMBER OF FILE LABEL WORDS * (including date and reel #) NUMBER OF FREE WORDS
04 DATE OF RECORDING (FIELDATA CODE)
05 LANGUAGE OF LABEL 0 - XS3 1 - FIELDATA REEL NUMBER (BINARY)
06
File Label
233 words for 240 word block
113 words for 120 word bloc
FREE WORDS
BALANCE OF BLOCK (IF ANY) DISREGARDED
1-2 BLOCK LENGTH - CURRENT FILE ITEM SIZE - CURRENT FILE
1-1 1 1 0 BLOCK COUNT USABLE TAPE
1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1

* SORT will typeout up to 21 words for this LION INTERNAL format.

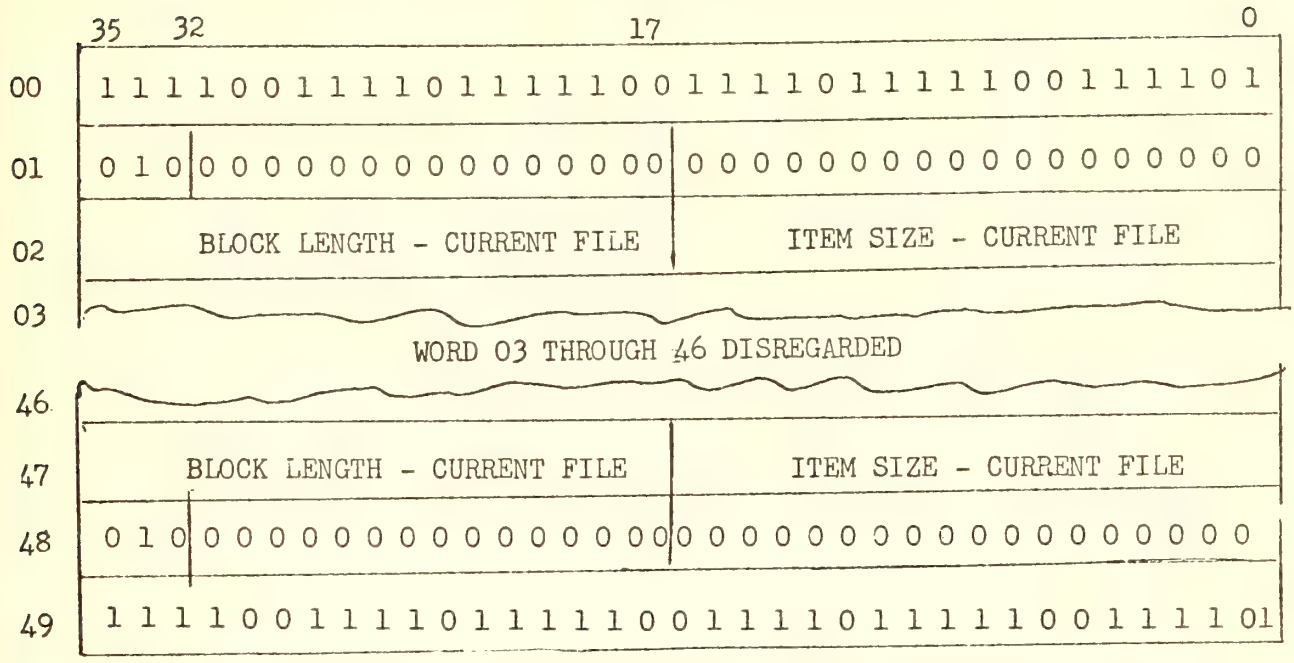
FIXED LENGTH ITEM BLOCK (LION INTERNAL)



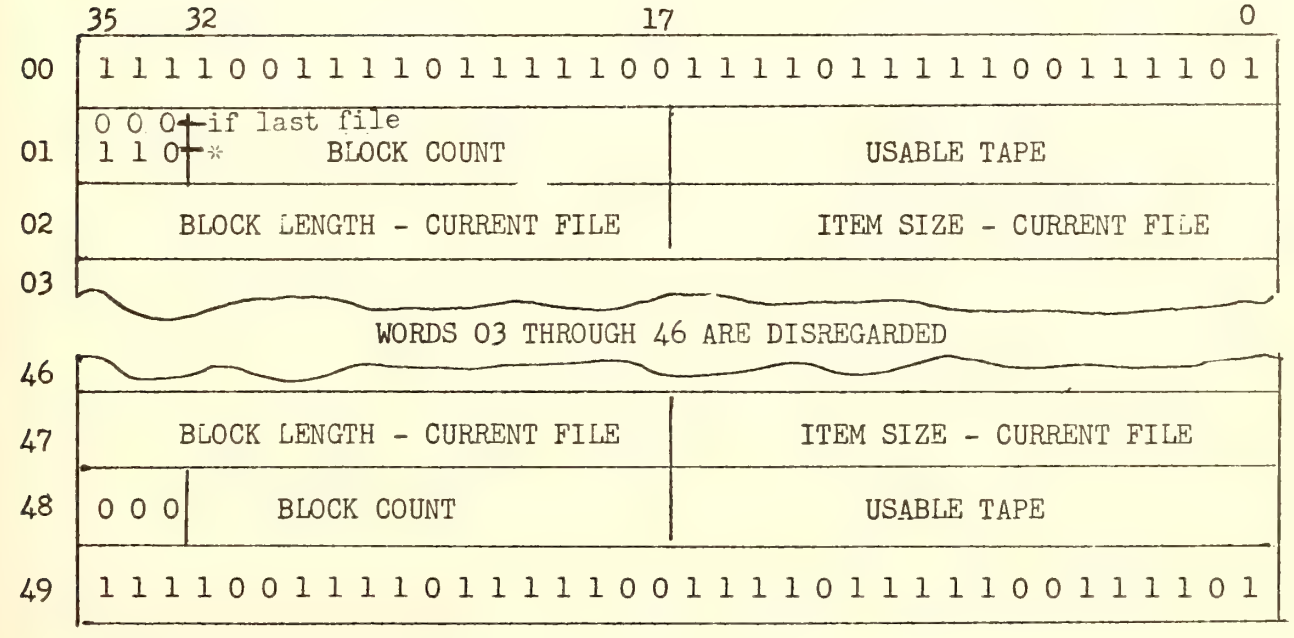
VARIABLE LENGTH ITEM BLOCK (LION INTERNAL)


35
17
0
NO. ITEMS IN BLOCK
NO. WORDS IN BLOCK
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NO.WORDS,ITEM FOLLOWING
DATA ITEM (VARIABLE LENGTH)
NO. WORDS,ITEM PRECEDING
NO.WORDS,ITEM FOLLOWING
Item-Count Words
NO.WORDS,ITEM PRECEDING
NO. WORDS, ITEM FOLLOWING
DATA ITEM (VARIABLE LENGTH)
NO.WORDS,ITEM PRECEDING
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BALANCE(IF ANY)DISREGARDED
NO.WORDS DISGREGARDED
CHECKSUM
NO.ITEM IN BLOCK
NO.WORDS IN BLOCK

END OF REEL SENTINEL BLOCK (LION INTERNAL)



END OF FILE SENTINEL BLOCK (LION INTERNAL)


35
32
17
0
00
1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1
01
0 0 0 if last file
1 1 0 *
BLOCK COUNT
USABLE TAPE
02
BLOCK LENGTH - CURRENT FILE
ITEM SIZE - CURRENT FILE
03
WORDS 03 THROUGH 46 ARE DISREGARDED
46
47
BLOCK LENGTH - CURRENT FILE
ITEM SIZE - CURRENT FILE
48
0 0 0
BLOCK COUNT
USABLE TAPE
49
1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1

*if another file follows

LABEL BLOCK (IIA COMPATIBLE)

Word		
00	747474747474	
01	747474747474	
02	Label	Up to 63 words may be included
03	Label	in file label.(SORT uses only 4)
04	Label	
05	Label	Balance of block through word 118
.		"Z" filled. (octal 747474747474)
.		
.		
118	747474747474	
119	000000000060	Printer stop in last word

DATA BLOCK (IIA COMPATIBLE)

Record 1 Record 2 Record 3 Record 4 etc.

Record Size - must be a factor of 120 but greater than 2

Block Size - fixed at 120 words

TERMINAL BLOCKS (IIA COMPATIBLE)

When items do not fill the block, the block will be filled with terminal records. In the event that the last data record completes a block, the next block is filled with terminal records.

TERMINAL RECORD

00	747474747474	When records are less than 6 words,
01	747474747474	first 3 words only.
02	60xxxxxxxxxx	xxxxxxxxxx (Record count in XS3)
03	000000000000	
04	000000000000	
05	xxxxxxxxxxxx	xxxxxxxxxxxx (Checksum in XS3 formed from
Record Fill -	747474747474	the sum of left and right half sum of every data word)

06 - 119 terminal records

Last block on tape (120 words of "Z's")

Note - The input terminal record may consist of all "Z's". If the input checksum location consists of "Z's" or the constant "END P ", it is accepted.

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
0	35-00	FILE NAME (IN FIELDATA)	PARAMETER 1 OF IOCON	PARAMETER 1 (A) OF OPNOUT
1	35-18	ADDRESS OF LABEL	PARAMETER 2 OF IOCON	PARAMETER 7 (G) OF OPNOUT
1	17-00	NUMBER OF LABEL WORDS	" " "	FIRST WORD OF LABEL
2	35-18	IIIA AND DRUM - ADDRESS OF FREE WORDS	PARAMETER 3 OF IOCON	NOT USED
		IIA - ITEMS YET TO BE WRITTEN TO FILL BUFFER	SUPPLIED BY LION	SUPPLIED BY FOLION
2	17-00	IIIA AND DRUM - NUMBER OF FREE WORDS	PARAMETER 3 OF IOCON	NOT USED
		IIA - NUMBER OF ITEMS PER BUFFER	SUPPLIED BY LION	SUPPLIED BY FOLION
3	35-30	FILE FLAG 1 = IIA 3 = IIIA 6 = DRUM	PARAMETER 4 OF IOCON	SUPPLIED BY FOLION

FILE CONTROL TABLE -- OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
3	29-00	CHANNEL AND UNIT NUMBER DRUM - STARTING ADDRESS	PARAMETER 4 OF IOCON	PARAMETER 5 (E) OF OPNOUT
4	35-00	ALTERNATE CHANNEL AND UNIT	PARAMETER 5 OF IOCON	PARAMETER 6 (F) OF OPNOUT
		DRUM - CURRENT ADDRESS	SUPPLIED BY LION	SUPPLIED BY FOLION
5	35-18	END OF REEL EXIT	PARAMETER 9 OF IOCON	PARAMETER 9 (I) OF OPNOUT
5	17-00	FILE CONTROL ERROR EXIT	PARAMETER 10 OF IOCON	PARAMETER 8 (H) OF OPNOUT
6	35-18	WRITE ERROR EXIT	PARAMETER 11 OF IOCON	SUPPLIED BY FOLION
6	17-00	INITIAL ADDRESS CURRENT BUFFER	SUPPLIED BY LION	SUPPLIED BY FOLION
NOTE:	WORDS	7-11 ARE I/O EXECUTION PACKET FOR	BUFFER #1	
7	35-30	STATUS CODE	EXEC	EXEC
7	29-24	INTERRUPT CODE	EXEC	EXEC
7	22-00	USED BY DRUM WRITE ROUTINE (SEE EXEC MANUAL)	EXEC	EXEC

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
8	35-30	EXEC I/O FUNCTION CODE	SUPPLIED BY LION	SUPPLIED BY FOLION
8	29-00	CHANNEL AND UNIT NUMBER DRUM - STARTING ADDRESS FOR WRITE	" " " " " "	" " " " " "
9	35-34	TRANSFER MODE DESIGNATOR	" " "	" " "
9	33-18	NUMBER OF WORDS TO WRITE	" " "	" " "
9	17-00	ADDRESS OF BUFFER #1	" " "	" " "
10	35-34	TRANSFER MODE DESIGNATOR	" " "	" " "
10	33-18	NUMBER OF WORDS IN BUFFER NOT WRITTEN	EXEC	EXEC
10	17-00	BUFFER ADDRESS FROM WHICH LAST WORD WAS WRITTEN	EXEC	EXEC

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
11	35-18	NUMBER OF BLOCKS READ (FOR BACKWARD READ OF PREVIOUS END OF FILE)	EXEC	EXEC
11	17-00	NUMBER OF BLOCKS TO READ (BACKWARDS)	SUPPLIED BY LION	SUPPLIED BY FOLION
12 1/	35-30	DRUM FLAG "01 = DRUM	" " "	" " "
12	29-24	"01" = FILE CLOSED	" " "	" " "
12	23-18	NUMBER OF BLOCKS TO BE REWRITTEN ON NEW REEL (WHEN A WRITE DETECTS END OF TAPE)	" " "	" " "
12	17-12	"01" = END OF REEL	" " "	" " "
12 2/	11-06	END OF REEL REWIND FUNCTION	" " "	" " "
12	05-00	"01" = FILE CLOSED PREMATURELY (BEFORE END OF REEL)	" " "	" " "

1/ $H1 CONTAINS 747474_8 AFTER OENFIL EXECUTED (IIA ONLY)

2/ $T3 CONTAINS NUMBER OF WORDS WRITTEN ON LAST WRITE OF FINAL BUFFER (IIA ONLY)

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
NOTE:		WORDS 13-16 ARE I/O EXECUTION PACKET FOR BUFFER #2 (USED BY LION ONLY). CONTENTS ARE THE SAME AS WORDS 7-10 EXCEPT THAT ALTERNATE OUTPUT UNIT IS USED (IF CODED) AND ALL OTHER INFORMATION PERTAINS TO OUTPUT OPERATIONS ON BUFFER #2.		
		THE FOLLOWING (WORDS 13-16) PERTAINS TO FOLION ONLY:		
13	35-18	NOT USED	SEE NOTE ABOVE	
13	17-12	"01" = FOLION	" " "	SUPPLIED BY FOLION
13	11-00	VERSION NUMBER (IIIA AND DRUM)	" " "	" " "
14	35-30	IIIA - FILE FLAG	" " "	" " "
		IIA - I/O FUNCTION	" " "	" " "
14	29-00	TAPE - CHANNEL AND UNIT NUMBER	" " "	" " "
15	35-00	NOT USED		
16	35-00	NOT USED		

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
17	35-06	NOT USED		
17	05-00	IIIA AND DRUM -		
		"01" - BUFFER ≥ 240 WORDS	SUPPLIED BY LION	SUPPLIED BY FOLION
		"00" - BUFFER < 240 WORDS		
18	35-18	IIA - NUMBER OF DATA BLOCKS WRITTEN	" " "	" " "
18	17-00	CURRENT BUFFER ADDRESS	" " "	" " "
19	35-18	BUFFER SIZE	PARAMETER 8 OF IOCON	PARAMETER 3 (C) OF OPNOUT
19	17-00	ITEM LENGTH ("0" FOR VARIABLE LENGTH)	PARAMETER 12 OF IOCON	PARAMETER 4 (D) OF OPNOUT
20	35-00	IIIA - BLOCK COUNT (AVAILABLE TAPE)	SUPPLIED BY LION	SUPPLIED BY FOLION (SET TO "0" AFTER FILE CLOSED)
		IIA - MAXIMUM NUMBER OF BLOCKS TO WRITE	PARAMETER OF IOCON	" " "
		DRUM - AVAILABLE DRUM ROOM	SUPPLIED BY LION	" " "

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
*21	35-00	IIA - CHECKSUM STORAGE	SUPPLIED BY LION	SUPPLIED BY FOLION
22	35-00	IIA - STORAGE FOR BUFFER STARTING ADDRESS	" " "	" " "
23	29-24	IIIA AND DRUM - "01" = FILE OPENED	" " "	" " "
24	35-00	SINGLE BUFFER OPERATIONS - BOTH HALVES CONTAIN ADDRESS OF WORD 7. FOR DOUBLE BUFFER ONE HALD WILL CONTAIN THE ADDRESS OF WORD 13	" " "	" " "
25	35-00	NOT USED	" " "	" " "
26	11-6	"01" = FILE OPENED	" " "	" " "
27	35-00	NOT USED		
28	35-00	NOT USED		
*	WORDS 21-22	USED AS COMMUNICATIONS PACKET FOR CHANGE MESSAGE (IIIA ONLY)		

FILE CONTROL TABLE - OUTPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
29	35-18	END OF REEL RETURN	NOT USED BY LION	SUPPLIED BY FOLION
29	17-00	FILE CONTROL ERROR RETURN	" " " "	" " "
30	17-00	STORAGE FOR $B11 (FOR ITMRTN)	" " " "	" " "
31	35-00	CULULATIVE CHECKSUM (IIIA ONLY) AND DRUM	SUPPLIED BY LION	" " "
32	35-00	CULULATIVE RECORD COUNT (IIIA ONLY) AND DRUM	" " "	" " "
*33	35-00	NOT USED		
34	35-00	NOT USED		

* LION - WORDS 33 AND 34 DO NOT EXIST IN FILE CONTROL MACRO

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
0	35-00	FILE NAME (IN FIELDATA)	PARAMETER 1 OF IICON	PARAMETER 1 (A) OF OPNIN
1	35-18	ADDRESS OF LABEL	PARAMETER 2 OF IICON	PARAMETER 6 (F) OF OPNIN
1	17-00	NUMBER OF LABEL WORDS	PARAMETER 2 OF IICON	FIRST WORD OF LABEL
2	35-30	TAPE DIRECTION 0 = FORWARD 1 = BACKWARD	PARAMETER 3 OF IICON	PARAMETER 7 (G) OF OPNIN
2	29-18	ITEM SIZE (IIA ONLY)	SUPPLIED BY LION	SUPPLIED BY FOLION
2	17-00	IIIA AND DRUM - SEARCH, READ READ SENTINEL FUNCTION CODES	" " "	" " "
		IIA - NUMBER OF ITEMS PER BUFFER	" " "	" " "
3	35-30	FILE FLAG 1 = IIA 3 = IIIA 6 = DRUM	FIRST HALF OF PARAMETER 4 OF II CON	SUPPLIED BY FOLION

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
3	29-00	CHANNEL AND UNIT NUMBER DRUM - FIRST DRUM ADDRESS	PARAMETER 4 of IICON	PARAMETER 4 (D) OF OPNIN
4	35-00	SAME AS WORD 3 (FOR ALTERNATE INPUT DEVICE)	PARAMETER 5 OF IICON	PARAMETER 5 (E) OF OPNIN
		DRUM - CURRENT DRUM ADDRESS	SUPPLIED BY LION	SUPPLIED BY FOLION
5	35-18	IIIA ONLY - END OF REEL EXIT	PARAMETER 9 OF IICON	PARAMETER 9 (I) OF OPNIN
5	17-00	FILE CONTROL ERROR EXIT	PARAMETER 10 OF IICON	PARAMETER 8 (H) OF OPNIN
6	35-18	READ ERROR EXIT	PARAMETER 11 OF IICON	SUPPLIED BY FOLION
6	17-00	NOT USED		
NOTE: WORDS 7-11 ARE I/O EXECUTION PACKET FOR BUFFER #1				
7	35-30	STATUS CODE	EXEC	EXEC
7	29-24	INTERRUPT CODE	" 40 - Normal 34 - EOF	" See EXEC VII - 42
7	22-00	IIIA - BUFFER SIZE	SUPPLIED BY LION	SUPPLIED BY FOLION
		DRUM - USED BY DRUM READ ROUTINE (SEE EXEC MANUAL)		

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
8	35-30	EXEC I/O FUNCTION CODE	SUPPLIED BY LION	SUPPLIED BY FOLION
8	29-00	CHANNEL AND UNIT	SAME AS WORD 3	SAME AS WORD 3
		DRUM - ADDRESS AT WHICH DRUM READ IS TO START	SUPPLIED BY LION	SUPPLIED BY FOLION
9	35-34	TRANSFER MODE DESIGNATOR	SUPPLIED BY LION	SUPPLIED BY FOLION
9	33-18	NUMBER OF WORDS TO READ	SUPPLIED BY LION	SUPPLIED BY FOLION
9	17-00	ADDRESS INTO WHICH FIRST WORD IS TO BE READ	SUPPLIED BY LION	SUPPLIED BY FOLION
10	35-34	TRANSFER MODE DESIGNATOR	" " "	" " "
10	33-18	NUMBER OF WORDS	EXEC	EXEC
		TO READ LESS THE NUMBER OF WORDS ACTUALLY READ		
10	17-00	ADDRESS INTO WHICH LAST WORD WAS READ	EXEC	EXEC

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
11	35-18	NUMBER OF BLOCKS READ	EXEC	EXEC
11	17-00	" " " " TO READ	SUPPLIED BY LION	SUPPLIED BY FOLION
12	35-30	DRUM ONLY - "01" - DRUM FLAG	" " "	" " "
12	35-00	SENTINEL WORD	" " "	" " "
NOTE:		LION ONLY - WORDS 13-18 ARE I/O EXECUTION PACKET FOR BUFFER #2. THEIR CONTENTS ARE THE SAME AS WORDS 7-12 EXCEPT THAT THE ALTERNATE INPUT DEVICE IS USED (IF CODED) AND ALL INFORMATION PERTAINS TO I/O REQUESTS FOR BUFFER #2.		
		THE FOLLOWING INFORMATION (WORDS 13-18) APPLIED TO FOLION ONLY:		
13	35-18	NOT USED		
13	17-12	FOLION FLAG ("01" = FOLION)	SEE NOTE ABOVE	SUPPLIED BY FOLION
13	11-00	VERSION NUMBER (IIIA INPUT ONLY)	" " "	" " "

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
14	35-30	IIIA AND DRUM - FILE FLAG	SEE NOTE ABOVE	SUPPLIED BY FOLION
		IIA - FUNCTION CODE	" " "	" " "
14	29-00	CHANNEL AND UNIT DRUM - STARTING ADDRESS	" " "	" " "
15	35-18	IIA ONLY - BUFFER SIZE	" " "	" " "
16	35-00	NOT USED		
17	35-18	NOT USED		
17	17-00	NUMBER OF BLOCKS TO READ	" " "	" " "
18	35-00	SENTINEL WORD	" " "	" " "
19	35-18	BUFFER LENGTH	PARAMETER 8 OF IICON	PARAMETER 3 (C) OF OPNIN
19	17-00	ITEM SIZE	IIIA AND DRUM - SUPPLIED BY LION	IIIA AND DRUM - SUPPLIED BY FOLION
		("0" FOR VARIABLE LENGTH)	IIA - PARAMETER 3 OF IICON	PARAMETER 9 (I) OF OPNIN

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
20	35-00	IIIA - BLOCKS READ *		SUPPLIED BY FOLION
		DRUM - TABLE LENGTH (FOLION ONLY)		
20	35-00	IIIA - AVAILABLE TAPE	SUPPLIED BY LION	
		DRUM - TABLE LENGTH (LION ONLY)		
21 1/	35-00	IIA - TEMPORARY CHECKSUM STORAGE	" " "	" " "
22	35-18	IIA - CUMULATIVE ITEM COUNT	" " "	" " "
23	35-18	ITEMS IN BLOCK YET TO BE READ	" " "	" " "
23	17-00	IIIA - NUMBER OF DISREGARDED WORDS IN BLOCK	" " "	" " "

1/ WDS 21-24 ARE USED AS COMMUNICATIONS PACKET FOR CHANGE MESSAGES

* SET TO ZERO AFTER FILE CLOSED

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
24	35-00	FOR SINGLE BUFFER OPERATIONS BOTH HALVES WILL CONTAIN THE ADDRESS OF WORD 7. FOR DOUBLE BUFFER OPERATIONS ONE HALF WILL CONTAIN THE ADDRESS OF WORD 13.	SUPPLIED BY LION	SUPPLIED BY FOLION
25	35-30	IIIA AND DRUM - "01" = LABEL READ AND CHECKED	" " "	" " "
25	29-24	IIIA AND DRUM - "01" = ITEM BEING PROCESSED IS LAST IN BUFFER	" " "	" " "
25	23-18	IIIA AND DRUM - "01" = END OF FILE SENTINEL DETECTED	" " "	" " "
25	17-12	IIA - CONTAINS 53_8 IIIA - "01" = END OF REEL	" " "	" " "
25	11-06	IIIA AND DRUM - "01" = BYPASS DETECTED	" " "	" " "
25	05-00	"01" = END OF FILE REACHED	" " "	" " "

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
26	35-24	NOT USED	SUPPLIED BY LION	SUPPLIED BY FOLION
26	23-18	IIIA and DRUM - "01" = LABEL READ	" " "	" " "
26	17-12	IIIA - END OF REEL REWIND CODE	" " "	" " "
26	11-06	IIA - "01" = FILE CLOSED	" " "	" " "
26	05-00	IIIA - "01" = FILE CLOSED	" " "	" " "
NOTE: AFTER NORMAL END OF FILE PROCEDURE WORD 25 $S3 WILL CONTAIN "01". THE REMAINDER OF WORDS 25 AND 26 WILL BE "0" EXCEPT IN THE CASE OF MULTI REEL FILES WHEN $S4 OF WORD 26 WILL CONTAIN THE REWIND CODE.				
27	35-24	IIIA - NUMBER OF BLOCKS IN BUFFER ALREADY PROCESSED	SUPPLIED BY LION	SUPPLIED BY FOLION
27	23-18	"01" MEANS A READ ERROR HAS OCCURRED	" " "	" " "

FILE CONTROL TABLE - INPUT

WORD	BITS	CONTENTS	SOURCE (LION)	SOURCE (FOLION)
27	17-00	CURRENT BUFFER POSITION	SUPPLIED BY LION	SUPPLIED BY FOLION
28	35-00	NOT USED		
29	35-18	END OF REEL RETURN	NOT USED BY LION	SUPPLIED BY FOLION
29	17-00	FILE CONTROL ERROR RETURN	" " " "	" " "
30	35-18	ADDRESS OF CURRENT BLOCK CHECKSUM (IIIA ONLY)	" " " "	" " "
30	17-00	STORAGE FOR $B11 (FOR ITM RTN)	" " " "	" " "
31	35-00	COMPUTED CHECKSUM	SUPPLIED BY LION	" " "
32	35-00	COMPUTED RECORD COUNT	" " "	" " "
33	35-00	TAPE/FILE CHECKSUM	" " "	" " "
34	35-00	TAPE/FILE RECORD COUNT	" " "	" " "
NOTE: WORDS 31-34 ARE NOT USED BY IIA				

20. SLEUTH II

20.1. INTRODUCTION TO SLEUTH II

The SLEUTH II assembler may be called to assemble SLEUTH II* subroutines during the compilation of a FORTRAN program. The requirement is that an ASM card be used instead of a FOR card prior to the SLEUTH II coding. The form of the ASM card under the present system is:

@PI ASM $name_1$,$name_2$,$name_3$

where $name_1$, $name_2$, and $name_3$ stand for the source name, the updated source name, and the name attached to the relocatable binary output of the subroutine.

In order to write SLEUTH II coding, the programer should have a knowledge of the hardware characteristics of the UNIVAC 1107. It would be helpful to study the 1107 Census Computer Manual and SLEUTH I Programer's Reference Guide. The SLEUTH II Programer's Guide UP-3670, Revision 1,is the UNIVAC general manual.

This chapter is a short synopsis of the main elements in SLEUTH II. Appendix C lists the instruction repertoire.

20.2. INSTRUCTION WORD FORMAT

The format of the machine language word on a field basis is illustrated below:

	F	J	A	X	H	I	M
Bits	6	4	4	4	1	1	16

Where:

F indicates the operation code listed in Appendix C
J partial word determinant or minor operation code
A the register or I/O channel designation field
X the index register designation field (sometimes referred to as the B register)
H the index register incrementation field
I the indirect address designation field
M the base operand address field (often called the u field)

The format of a symbolic instruction is altered for convenience of programing. Commas are used to separate operand subfields. The basic line of coding is divided into three or fewer fields. They are the LABEL, OPERATION, and OPERAND fields. Each field may be divided into subfields. A subfield is an expression which is terminated by a comma, unless it is the last subfield in the field. In this case a space terminates both the subfield and the field.

*A SLEUTH II element may be the main program followed by FORTRAN subroutines or an entire program may be written in SLEUTH II.

20.2. INSTRUCTION WORD FORMAT (Continued)

For instructions which involve a control memory location, the following formats are appropriate:

LABEL	OP	OPERAND
	F	A,M,X,J
	F,J	A,M,X

For instructions which do not involve a control memory location, the following format is used.

LABEL	OP	OPERAND
	F	M,X

The entry in the F field is the instruction mnemonic. Unique mnemonics have been created for each operation code and sub-function code combination. Therefore, the entry in the J field will not be used for sub-function codes.

SLEUTH II expects the A field to represent the absolute film memory address of an arithmetic, index or R register, depending upon the instruction mnemonic. The following table supplies the addresses in thin-film memory where the registers are located

1-15 (octal 1-17) X registers --index registers sometimes called B registers. There are 15 index registers on thin-film available to the programer. They are 36 bits in length.

Each index register is divided into two portions; the right half 18 bits, or Q portion, and the left half 18 bits, or Δ portion. The right half (Q portion) is the address modifier. In operations specifying modification of the operand address by the index register, the Q portion is added to the base address shown in the M (or u) field before the operand is referenced. Thus the operand is taken from the modified address (or effective address). This Q portion is a signed number and may be + or -.

20.2. INSTRUCTION WORD FORMAT (Continued)

The Δ portion, the left-hand 18 bits, contains the increment which may be applied to the Q portion if the instruction specifies address incrementation after the Q portion has been used. Thus the incremented modifier will be used the next time that index register is used. Index register incrementation is indicated by preceding the X field with an asterisk.

12-27 (octal 14-33) A registers (overlap X 12-15)
There are 16 thin-film arithmetic registers. Each of these registers is 36 bits in length. They may be used for the accumulation of totals in arithmetic operations, both fixed and floating point. They are also used in load and store instructions, logical instructions, shifting, tests, and searches. Only the input/output and a few other instructions do not use an A register. Certain instructions use two consecutive A registers: Product of a multiplication, dividend of a division, certain shift instructions, and floating point instructions.

The assembler will subtract 12 from the value of the entry for an A register.

64-79 (octal 100-117) R registers
There are 16 R registers on thin-film. The first four of these registers are special purpose registers. R64 (0) the real-time clock which is under control of EXEC. R65 (1) the repeat counter which the programer will load before executing a repeated instruction. R66 (2) is the mask register which the programer will load for certain instructions. R67 (3) is the temporary storage for the address of the next instruction. It is used during repeat instructions. The remainder of the R registers may be used as storage areas.

The assembler will subtract 64 from the value of the entry for an R register.

20.2. INSTRUCTION WORD FORMAT (Continued)

Registers and partial word determinants (J field) may be addressed by using mnemonic designators (which are predefined and equated within the Census SLEUTH II Assembler for the 1107).

Registers	Mnemonic	Address
X	X1 - X15 or B1 - B15	1-15
A	A0 - A15	12-27
Q	Q0 - Q3	Overlap X12-X15 properties of A and X
R	R1 - R15	64-79

Partial Word Determinants	Mnemonic	Octal Code
Left half	H1	2
Right half	H2	1
Left half, sign extension	XH1	4
Right half, sign extension	XH2	3
First third, sign extension	T1	7
Second third, sign extension	T2	6
Third third, sign extension	T3	5
Sixth of word	S1 - S6	15 - 10
Contents of M field is operand	U or M	16
Contents of M field with sign extension	XU or XM	17

EXAMPLE:

```
L  A0,10,,U    may be used instead of
L  12,10,,016
```

20.2. INSTRUCTION WORD FORMAT (Continued)

If the instruction mnemonic is such that the A field designates an A, X, or R register, the instruction may be coded to omit the A, X, or R and the A field is examined to determine the proper code. Thus, in the following examples, the resultant codes are equivalent.

```
L   17,M is equivalent to LA 17,M
L    2,M is equivalent to LX  2,M
L   65,M is equivalent to LR 65,M
```

The entry in the M field respresents the base operand address. Indirect addressing is indicated by preceding the M field with an asterisk.

The entry in the X field represents the specific index register to be used.

The entry in the J field is used to designate partial word transfers to and from the arithmetic unit.

20.3. THE LABEL FIELD

The label field in SLEUTH II may consist of a declaration of a specific location counter, a label, or a combination. If the latter is desired, the location counter declaration is the first entry on the symbolic line, and is followed by a comma if a label is also present. In any case, a space in the first position implies that none of the above is present. The existence of a label is indicated by the presence of a valid character (A-Z) in the first position of a symbolic line. A "$" in the first position signifies that a location counter is to be specified.

A. Location Counter Declaration

32 location counters are available in SLEUTH II; however, if no location counter is explicitly used, the program is controlled by location counter zero. The declaration of a specific counter is accomplished by entering $(e) as the first entry in the label field with "e" being a number between 0 and 31. These counters are used to regroup, in any arbitrary manner, lines of coding. It is useful in segmentation (See INFO directive). Each new location counter entry begins the coding relative to zero, and coding under a previously defined counter will continue at the last address specified for that counter. $(e) -- "e" may be an expression as well as a hard coded constant.

B. Labels

A label may be from 1 to 6 characters long, the first character must be pure alphabetic (A-Z). Succeeding character may be any combination -- alphabetics, numerics (0-9), or $. A label may be subscripted for uniqueness. The subscript does not count as an integral character of the label (i.e. CAT apart from CAT(1)). The subscript may also represent a dummy value within a procedure or function.

20.3. THE LABEL FIELD (Continued)

B. Labels (Continued)

An asterisk may be affixed to a label. When this label is defined outside a procedure, it becomes externally defined. This means the label is known outside of the program. The asterisk does not count as a character of the label. (Examples: CAT* or CAT*(1))

If a location counter is to precede a labelled line, a comma and the label must immediately follow the location counter designation.
(Example: $(1),CAT)

FORMAT IS $(e),LABEL ....

20.4. THE OPERATION FIELD

If the first character position of a coded line is blank, the line is considered not to possess a label. The first non-blank character following column one is interpreted to be the start of the OPERATION field (exception is a period, semicolon, or apostrophe).

The operation field may contain 4 types of entries:

1. An instruction mnemonic, with a possible j designation (may not be used if that instruction is contained in a literal).

2. + or -, indicating a data word; or single quotes may surround an alpha constant.

3. An assembler directive.

4. A label previously defined as a legitimate entry point to a PROC or FUNC.

In all the above cases, except 2 which is optional, a space following any character except a comma ends the OPERATION field.

Unless the directive RES is present, the controlling location counter will be incremented by one after each word is generated.

20.5. THE OPERAND FIELD

The OPERAND field may contain subfields which are separated by commas. Any operand may contain fewer than the maximum number of subfields, or none. If a subfield other than the normal first or last is to be omitted, two contiguous commas or a comma, zero, comma (,0,) is necessary. If the first normal subfield is to be omitted, a "," must be coded. A period space coded just after the last element will cause scanning to cease and will speed up assembly time.

20.5. THE OPERAND FIELD (Continued)

EXAMPLE:

```
$(3),FRD AND 17,TABL,1,4
        XOR 20,WS,,U
. The missing subfields in the second
. line have a value of zero to the assembler
```

20.6. CONTINUATION

If a semicolon (;) is encountered outside of an alphabetic item, the current line is continued with the first non-blank on the following line. Any characters on a line after the semicolon are considered to be comments.

20.7. TERMINATION

Further operand information is not interpreted when the maximum number of subfields required by the operation have been encountered, or by the assembler's recognition of 80 characters, whichever occurs first. A period space coded after the last subfield will cause scanning to cease and speed up assembly time.

Following the information portion of a line, any characters may be entered as comments.

Example:

```
    LN;
A   16, ;
TABL, 3,;
2. THIS WILL LOAD ARITHMETIC REGISTER A4
. WITH THE COMPLEMENTED LEFT HALF OF THE
. WORD ADDRESSED BY THE LABEL TABL AS
. MODIFIED BY INDEX 3  THIS EXPLANATION
. WOULD APPEAR AS A COMMENT LINE
```

20.8. DATA WORD GENERATION

A + or - in the operation field, followed by one or more subfields separated by commas in the operand field, may be used to generate a constant word. Optionally, the operand may follow immediately. An alpha constant such as 'ABC' coded without the "+" produces a left justified word in contrast to the following rule.

20.8. DATA WORD GENERATION (Continued)

If the operand contains one subfield, the value of the subfield will be right-justified in a 36-bit word. Two subfields result in two 18-bit fields. Three subfields result in three 12-bit fields. Six subfields create six 6-bit fields. Each subfield may be signed independently. Subfields may be decimal, octal (precede number by 0), or alpha (enclose in single quotes).

Examples:

```
-16384          . octal 777777737777
+ 'B', -0257 . octal 000007777520
-56,0407,-313. octal 770704077306
8,-04,21,-28,017,-14
. The above line produces octal 107325431761
```

20.9. EXPRESSIONS

An expression is an elementary item or a series of elementary items connected by operators. (See Operators.) Blanks are not permitted within an expression.

Elementary Items

An elementary item may be a label, a location, an octal number, a decimal number, an alphabetic, a floating point number, a line literal, or a parameter.

a. Any label may be used as an elementary item. Whenever a label is encountered within an expression, the value equated to the label is substituted for the label within the expression.

EXAMPLE:

```
CONST     EQU     010000  .
          LA,M    16,CONST .
```

b. The symbol for a current location counter reference is "$". When $+n is coded, care should be taken so the interval does not extend over a procedure call where the procedure is of variable length.

EXAMPLE:

```
J     $+2     . JUMP
LA    16,TEMP . TO
SA    16,WS   . THIS LINE
```

20.9. EXPRESSIONS (Continued)

Elementary Items (Continued)

c. Octal digits (0-7) may be represented as an elementary item by preceding the digits with a zero. The binary number will be right-justified in a signed field.

EXAMPLE:

```
+017 . produces octal 000000000017
-074 . produces octal 777777777703
```

d. Decimal values may appear as an elementary item within an expression. The binary number will be right-justified in a signed field. The number must not be preceded by zero.

EXAMPLE:

```
+12  . produces octal 000000000014
-12  . produces octal 777777777763
```

e. Alphabetics may be represented by enclosing the alphabetic characters with apostrophes. If a + precedes the term, then the value will be right-justified with zero fill to the left. If no +, then the value will be left-justified with Fieldata spacefill to the right. An alphabetic item used as a literal is assumed to have the plus. If a plus precedes the item, the number of characters is limited to six. The CHAR declarative may alter the alphabetic characters.

EXAMPLE:

```
+ 'HEAD'  . will produce octal 000015120611
'HEAD'    . will produce octal 151206110505
```

f. Floating Point Numbers are represented by placing a decimal point in the value. The decimal point must be preceded and followed by at least one digit. The value will be represented in the UNIVAC 1107 internal floating point format.

EXAMPLE:

```
+16384.0  . floating point 217400000000
```

20.9. EXPRESSIONS (Continued)

Elementary Items (Continued)

g. A literal may be represented on an instruction line by enclosing the constant within parentheses in the normal M portion of the instruction word. This will cause the assembler to generate a word containing the constant to be sent to the end of the program and duplicates will be eliminated for a particular counter.

In addition to a constant data word, the line item may be an instruction word. A label is not permitted in the case of an instruction word line item. The first character following the left parenthesis must be the start of the OPERATION field.

EXAMPLES OF LINE ITEMS:

```
LA    16, (0400)     . octal 400
TE    17, ('BRING')  . alpha 'BRING'
TNE   18, (J RAT)    . instruction word of J RAT
LA    19,(((899))) . the location of the location of
                     . the constant 899 will be
                     . loaded into A7
 A    14, (TE 16,(3)) . two literals will be generated
```

h. An elementary item may be a PROC or FUNC parameter. See PROC and FUNC descriptions.

20.10. OPERATORS

There are 14 operators which designate the method and sequence of combining elementary items or expressions with a subfield. Blanks are not permitted within an expression. Evaluation of an expression begins with substitution for each element. The operations are then performed from left to right in order of hierarchy. Parentheses may be used with care to alter the order of operations. Note that these are assembly time operators. The value produced by operators should not be confused with the value produced by a literal and therefore is not an address. The order of hierarchy is the highest number first.

20.10. OPERATORS (Continued)

HIERARCHY	OPERATOR	DESCRIPTION
6	*+	a*+b is equivalent to $a*10^{b}$
	*-	a*-b is equivalent to $a*10^{-b}$
	*/	a*/b is equivalent to $a*2^{b}$
5	*	arithmetic product
	/	arithmetic quotient
	//	covered quotient (a//b is equivalent to $\left(\frac{a+b-1}{b}\right)$)
4	+	arithmetic sum
	-	arithmetic difference
3	**	logical product (AND)
2	++	logical sum (OR)
	--	logical difference (Exclusive OR)
1	=	a=b has the value 1 if true, 0 if otherwise
	>	a>b has the value 1 if true, 0 if otherwise
	<	a<b has the value 1 if true, 0 if otherwise

20.10. OPERATORS (Continued)

EXAMPLES Of Operators:

a. = : The assembler compares the value of two items or expressions and assigns a value of 1 if equal, a value of 0 if otherwise.

```
 DO A=3, RES 3
. If the condition specified is met the controlling
. location counter will be incremented by 3
. otherwise the line will be skipped
```

b. > : The assembler makes the comparison between two items. If the value of the first expression is greater than the second, the value of the expression will be 1, otherwise it will be 0.

```
(A>2)*5
. If A is >than 2 the value of the
. expression is 5, otherwise the
. expression value is 0.
```

c. < : The test in this case is less than.

```
 +A<2*2
· If A is<4 the expression value is 1
· otherwise the value is 0
```

d. ++: The assembler will produce the logical sum of the items or expressions and use the logical sum as the value of the expression.

```
A  EQU  4
   (A**1=0)* A++1
.  The value of the expression
.  above is five
```

e. --: The logical difference (EXCLUSIVE OR) produces the logical difference between two expressions or items.

```
A  EQU  3
   (A**1)*A--1
.  The value of the expression
.  above is two
```

20.10. OPERATORS (Continued)

EXAMPLES Of Operators:

f. ** : The logical product operator (AND) produces the logical product of the values of two expressions.

```
N EQU  17
  +N**3
. The value of the expression above
. is one
```

g. + : The arithmetic sum operator produces the algebraic sum of two items or expressions.

```
LX  8,WS+1
. Index register 8 will be loaded with
. contents of the word following the
. word labelled WS
```

h. - : The arithmetic difference operator produces the algebraic difference between the values of two items or expressions.

```
SA   16,AN-1
. The contents of register A4 will be stored
. in the word preceding the word
. labelled AN
```

i. * : The value of the expression is the product obtained by the multiplication of the two items.

```
N   EQU  17
    N-N/4*4
.   The value of the expression above
.   is one
```

j. / : The first expression is the dividend, the second is the division. The result is the quotient. The remainder is discarded.

```
B  EQU  16
  (B**3=0) *B/4
.  The value of the expression above
.  is four
```

20.10. OPERATORS (Continued)

EXAMPLES Of Operators:

k. // : The covered quotient operates in the same fashion as / (the arithmetic quotient) with the exception that if there is a remainder, the quotient is increased by 1.

```
A   EQU   3
    (A**370)*A//2
.   The value of the expression
.   above is two
```

l. *+ : The positive decimal exponent is a method of symbolically creating a floating point constant in UNIVAC 1107 format. a*+b is equivalent to $a*10^{b}$.

```
+0.234*+6
.  The value of the expression above is
.  octal 222711017776
```

m. *- : The negative decimal exponent functions in much the same manner as the positive exponent. a*-b is equivalent to $a*10^{-b}$.

```
+0.972*-3
.  The octal value of the expression
.  above is 166775467206
```

n. */ : The shift exponent allows the programer to enter a number and specify its binary positioning to the assembler. The shift may be left or right according to the sign of the exponent (-b will produce a right shift). a*/b is equivalent to $a*2^{b}$.

```
Z  DO 36, +0400000000000*/(1-Z)
.  The line above will generate 36 words
.  The octal value of the first word is
.  400000000000
.  The octal value of the last word is
.  777777777777
```

20.11. SLEUTH II ASSEMBLER DIRECTIVES

The symbolic assembler directives within SLEUTH II control or direct the assembly processor. These directives are represented by mnemonics which are written in the operation field of a symbolic line of code. The flexibility of each of these directives is the key to the power of the assembler. The directives are used to equate expressions, to adjust the location counter value, and to afford the programer special controls over the generation of object coding.

Before the directives within SLEUTH II are defined, it will be necessary to have a further discussion of labels and their areas of existence. As explained before, a label defined in a program proper (outside a procedure or function) is known only within the confines of the program. If an asterisk is affixed to the label, it becomes known outside the program (externally defined).

The procedure which will be more clearly defined under PROC can be thought of as a group of lines of symbolic coding independent from the program proper. The level of the procedure is considered to be one higher than the program, or one higher than the procedure within which it is nested. Labels defined within a procedure definition are known only in the procedure unless an asterisk is affixed. The asterisk results in the label being "lowered" a level and is then known to be available or recognizable to the program or to the next lower procedure within which it is nested. Labels in the program are always available to a higher level area. Similarly, labels within a procedure are available to the procedures which are nested within the procedure.

The directives are:

EQU	DO	WRD
RES	PROC	CHAR
FORM	NAME	LIST, UNLIST
END	GO	EQUF
LIT	FUNC	
INFO		

20.11.1. EQU

The EQU (EQUal) directive equates a label appearing in the label field to the value of the expression in the operand field.

FORMAT: LABEL EQU e_1

This value may be referenced in any succeeding line by use of the label equated to it. If a label is to be assigned a value by the programer, it must appear in an EQU line before it is considered defined.

20.11.1. EQU (Continued)

If a particular expression is used throughout a program or procedure, it is highly expeditious to use the EQU directive and substitute a simple label for the entire expression.

```
A   EQU   7/22816*31+ (5//B)
    LA    16,2*A,,U
```

When procedures are nested, labels which are defined in higher levels by EQU directives may be made available in outer procedures by affixing an asterisk in the innermost procedure in which it is defined.

```
L   EQU  010000
A4  EQU   16
    LA    A4,L,,U
. The load instruction will produce
. this object coding   10 16 04 00 0 010000
```

20.11.2. RES

The RES (REServe) directive allows a change to be made to the control counter by incrementation or decrementation. The operand field contains a value that may be represented through the use of any expression.

FORMAT: LABEL RES e_1

The RES directive may be used to create work areas for data or to specify absolute location counter positioning to the assembler. If a label is placed on the RES line, it is equated to the present value of the control counter, which is in effect the address of the first reserved word.

```
BETA  RES  010
      SA   16,BETA+4
. The line above will store the contents of
. register A4 in the fifth word of the
. reserved area BETA
```

20.11.3. FORM

The FORM directive is a means of describing a special word format designed by the user. This word format may comprise fields of variable length (within a word). The length in bits of each field is defined by the user through expressions in the operand field of a FORM line. The value of each expression specifies the number of bits desired in its respective field.

FORMAT: LABEL FORM e_1, ---, e_n

The number of bits specified by the sum of the values of the operand expressions cannot exceed 36.* The assembler uses the values of the operand expressions within the FORM line to create a control pattern that dictates a word format.

A reference to a word format is accomplished by writing the label of the FORM directive in the operation field followed by a series of expressions in the operand field which specify the value to be inserted in each field of a generated word. A reference to a specific FORM label will always create a word composed of fields in the same format. Of course, the contents of the fields may vary according to the expression values in the referencing line. Truncation will occur and an error flag set if a given value exceeds space permitted for a field as indicated in the FORM directive. If fewer or more than 36 bits are specified, no error flag will be set.

```
INSTR     FORM  6,4,4,4,4,2,16
          INSTR 054,0,04,01,0,010002
. The line above would produce an
. edited word as follows:
. 54 00 04 01 0 010002
```

20.11.4. END

The END directive indicates to SLEUTH II that it has reached the end of a logical sequence of coding. In the case of an END directive which terminates a program, the operand field should contain an expression which specifies the starting address of the program. In the case of a procedure, the operand field is ignored. In the case of a function, the expression in the operand field represents the value of the function. There may be no label associated with the END line.

FORMAT: END e_1

*This limit of 36 applies only to the 1107 word size.

20.11.5. LIT

The LIT (LITeral) directive defines a literal table under the control of the location counter in use when this directive is encountered by the assembler. Only one LIT directive is allowed for each location counter. Through the use of LIT directives, a number of separate literal tables can be created. Duplicate literals are eliminated within each unique literal table; however, duplicates may exist in separate literal tables. In the absence of a LIT directive, all literals will be placed in the literal table under control of location counter zero. The entries in the label field of a LIT directive comply with the rules of labeling concerning the location counter declaration and label construction. The label, however, may not be subscripted, be affixed by an asterisk, or be referenced.

```
 LA  16,(04)
. The octal literal 000000000004 will be
. placed in the literal table
. controlled by location counter zero
```

Use of the label with a literal will place the literal generated in the table of literals associated with the control counter current at the time the related LIT directive was encountered. The origin of the literal table follows the last coding line of the specified location counter. Duplicate literals are discarded in each table but may exist in separate literal tables.

If a literal table not under control of location zero is required, a LIT directive is used. If a LIT directive has no label, a literal not preceded by a label will be placed in the unlabeled literal table. If desired, unlabeled literals could be made to follow each program segment for which a separate location counter is used. If a LIT directive has a label, all literals to be placed in this literal table must be preceded by the label associated with this LIT directive.

FORMAT: LABEL LIT

REFERENCE: Label (literal)

EXAMPLES:

```
$(2)     LIT
   LA     16,(04)
. The octal literal 000000000004 will
. be placed in the literal table
. controlled by location counter two
```

20.11.5. LIT (Continued)

EXAMPLES: (Continued)

```
$(2),TOM LIT
        LA  16,(04)
        LX  3,TOM(01000)
. The literal 000000000004 will be placed
. in the literal table controlled by location
. counter zero
. The literal 000000001000 will be placed
. in the literal table controlled by
. location counter two
```

20.11.6. INFO

The INFO directive can be used to specify the sequence of location counters and their bank placement to the monitor system. An INFO directive is not necessary for program assembly. Without an INFO directive, the even-numbered declared location counters will be assigned sequential addresses in bank two and the odd-numbered location counters in bank one.

FORMAT: LABEL INFO a $C_1,C_2,....C_n$

The label is optional.
"a" represents the group number for type of storage.
Each of the C_n values will correspond to the location counters and labels $(Cn) coded in the program.

The group number has the following meanings:

0	The group will be absolute and no relocation will occur.
1,2	Bank one or bank two on a dependent basis. This pertains to segments which may be overlayed by previous segments.
5,6	Bank one or bank two on an independent basis.
3,7	Drum space is made available for the group, dependent or independent.
4	Common blocks.
33,34,37,38	Block Data.

```
    INFO    2    1,10,5,28,30
. The Line Above Will Place The Segments
. Defined By Location Counters 1,10,5,28,30
. In Bank Two In The Sequence Given
. By The Directive
```

20.11.7. DO

The DO directive is used to generate a specified value or line of coding a defined number of times. Two entries appear in the operand field of this directive. The second operand entry may be any valid symbolic line with or without a label. The number of times this line will be produced is determined by the value of the expression contained in the first operand entry. The two operand entries are separated by blank comma (Δ,). If there are no intervening blanks between the comma and the first character of the second operand entry, the symbolic line to be produced is assumed to have a label.

FORMAT: LABEL DO e_1Δ,ΔLINE OF CODING
or FORMAT: LABEL DO e_1Δ,LABEL ΔLINE OF CODING

A label may be written in the label field of the DO directive. In this case, the label is not equated to the location counter value, but to a counter whose initial value is always one. Each time the directive is executed this counter is incremented by one until the required limit, specified by the first operand expression, is reached.

```
I  DO  10 Δ, Δ+I
. THE VALUE I WILL BE GENERATED TEN
. TIMES THE FIRST VALUE OF I WILL
. BE 1  THE LAST VALUE WILL BE 10
```

If the number of times the DO is to be executed is negative, an E flag will be set and no lines generated.

The DO statement may be a conditional statement. That is, the number of times it is executed may be dependent on previously assigned values being altered.

An example is:

```
ΔDO  A<B Δ,Δ-3
. ASSUME A AND B HAVE VALUES ASSIGNED BY EQU STATEMENTS
. IF A<B IS TRUE THEN ONE LINE WILL BE PRODUCED
. OTHERWISE NO LINES WILL BE PRODUCED
.
```

```
A EQU 6
I DO   ((A**7)-6)+4 ,TAG(I)  +I* 2
  LA    16, TAG(2)
  LA    17, TAG(4)
.  THE DO LINE WILL CAUSE 4LINES TO BE GENERATED
.  HAVING ASSOCIATED LABELS OF TAG(1) TO TAG(4)
.  The CONTENTS OF THE LINES WILL BE 2,4,6,and 8
```

20.11.7. DO (Continued)

DO statements may be nested within DO statements up to a level of 8. As expected, the inner DO statements are executed first.

```
I  DO 8 , J DO  3 , +I+J
.
. THESE NESTED DO Statements will produce
. A TABLE OF 24 ENTRIES, THE CONTENTS OF
. WHICH ARE : 2,3,4,3,4,5,4,5,6,5,6,7,6,7,8,
. 7,8,9,8,9,10,9,10, AND 11
```

20.11.8. PROC

Often the programer will find a recurrence of sequences of coding very similar in nature. A device which permits the programer to generate similar sequences of coding is called the procedure. When a procedure sample in the source coding is encountered, the assembler stores all the lines of coding in the procedure and will generate this coding in the source program when the procedure is called upon. By varying the calling sequence, the assembler will modify the lines generated at assembly time.

The procedure definition is introduced into the source program by the PROC directive which must have a label associated with it. This label may be externally defined (using an *). The PROC line may or may not have an operand. The procedure itself is terminated by an END directive. If an operand is present on the PROC line, it will have a format of A,B.

```
BLAP*     PROC
            .
            .
            .
            .
            .
            .
          END
```

20.11.8. PROC (Continued)

Future calls on this procedure can be made by referencing the procedure label.

FORMAT OF PROCEDURE CALL

```
     BLAP
```

The procedure must be defined prior to a procedure call.

Many PROCS are written in such a way that the PROC label is not externally defined (does not have an *). In these cases, the PROC is referenced by other externally defined labels within it.

FIELDS, SUBFIELDS

In order to activate a given procedure, certain information must be given to the assembler at the time of call. The basic element of information supplied in the call is called a subfield. A string of subfields separated by commas is called a field. Fields are separated by spaces. Information given in these fields is transferred to the procedure sample into parameter reference forms which are subscripted indicators. For brevity, these parameter reference forms will be called paraforms.

The operand expressions of a procedure reference line provide specific values to be inserted into a general framework of coding. The expression ADDP(a,b) is an example of a dummy value in the procedure named ADDP. The assembler will look for the bth subfield of the ath field and substitute this value when the procedure is being evaluated.

An example of a PROC, a reference, and generated coding:

```
ADDP*   PROC                    . PROC
        LA      A4,  (ADDP(1,1))
        AA      A4,  CONSTA
        SS      A4,  ADDP(1,2)
        END

        ADDP     063,RAM        . REFERENCE

        LA      A4, (063)       . GENERATED CODING
        AA      A4, CONSTA
        SS      A4, RAM
```

In the above example, two subfields in one field were expected by the PROC. If the originator of the PROC wished to have 2 fields with 1 subfield each, the notations within the PROC could have been ADDP(1,1) and ADDP(2,1) and the calling line would have been:

```
ADDP    063    RAM
```

20.11.8. PROC (Continued)

OPERAND FIELD OF PROC LINE

A, the maximum number of fields furnished to the procedure, and B, the number of lines generated give the assembler enough information to know when to stop scanning input lines and to avoid double evaluation of the procedure.

It is essential that the B term be omitted if any of the following situations occur:

a. Forward references are made in the procedure.

b. External definitions are made in the procedure (except entry points).

c. The procedure could generate a variable number of lines.

d. When a change of location counter control occurs within a procedure.

An example of how a normal instruction word may be generated via a systems procedure is shown here for the LA instruction.

```
LOAD        PROC
LA*         NAME   010
F           FORM    6,4,4,4,2,16
            F      LOAD(0,0), LOAD(0,1)+LOAD(1,4), ;
LOAD(1,1)  -12,LOAD(1,3), 2*LOAD(1,*3) + LOAD(1,*2), ;
LOAD(1,2)
```

LOAD(0,0) is the value (010) on the name line which is the function code of the instruction LA.

LOAD(0,1) and LOAD(1,4) are the two arbitrary positions of the J designator.
LOAD(1,1) is the expected A designation.
LOAD(1,3) is the index designation.LOAD(1,*3) will set A 0 or 1 BIT in the incrementation designator field.
LOAD(1,*2) will set A 0 or 1 BIT in indirect addressing field.
LOAD(1,2) is the M(or U) portion of the instruction.

20.11.8. PROC (Continued)

Parameter Reference FORMS or PARAFORMS

L is the label on PROC or NAME line.
L - The number of fields submitted on call. If entry was made via NAME line, this figure is greater by 1.
L(a) - The number of subfields in the ath field (in the case of the FUNC it always refers to the number of subfields in the first and only list).
L(0,0) - operand on NAME line (meaningless if entry was not made via NAME).
L(0,1) - Second subfield of operation field submitted with a NAME directive, given at time of call.
L(0,a) - (a+1)th subfield of operation field submitted with a NAME directive.
L(n,M) - m th subfield, nth field of input information of call operand.
L(n,*M) - equal to 1 if mth subfield of nth field is preceded by asterisk; equal to Ø if not.

```
LABEL1*     PROC
LABEL2*     NAME  W
            .
            END
            . REFERENCES
            LABEL1 A,B C D E
            LABEL2,F,G N,O,P
```

(1) Entry Via LABEL1

LABEL1 = 4 (number of fields)
LABEL1 (1) = 2 (number of subfields in 1st field)
LABEL1 (1,1) = A
LABEL1 (2,1) = C
LABEL1 (3,1) = D
LABEL1 (4,1) = E

(2) Entry Via LABEL2

LABEL1 = (number of fields including NAME field) = 2
LABEL1 (0,0) = W
LABEL1 (0,1) = F
LABEL1 (3,1) = Ø (meaningless)

20.11.8. PROC (Continued)

Labels On A Reference Line

A label may be affixed to the line of reference to a procedure. Under normal conditions, this label will become associated with the first line of coding generated.

Example:

```
X *  PROC   1,2     . ONE LIST AND TWO LINES
. OF CODING TO BE GENERATED
     TLEM  X(1,1),4,,S3
     J   $+3
     END
     .
     .
RAM  X     A2              . TWO LINES GENERATED
     LA    A5,RAM          . RAM IS FIRST LINE
```

It is possible to associate a label with a line within the procedure other than the first one. This is done by coding an asterisk (*) alone in the label field of that particular line in the procedure definition area.

In the above example, if an asterisk had been coded in the label field of the line "J $+3", then this line would have been the one referenced as RAM.

Nested Procedures

The nesting of procedures can take two forms:

1. If a procedure definition is wholly contained within another procedure definition, it is explicitly nested in the larger and the internal procedure is considered to be one level higher than the procedure which contains it. If no other procedure bounds it, this procedure is considered to be one level higher than the externally bound procedure. This type of procedure may contain other procedures. A level of 63 is maximum. Entrances to internal procedures can be made only through its family of procedures, and never outside the external procedure unless extra asterisks are added to the internal-procedure entry points to raise their level. An internal procedure may only be referenced after a call has been made on the external procedure.

20.11.8. PROC (Continued)

Nested Procedures (Continued)

2. A procedure which is called upon by another is said to be nested within the calling procedure at the time of reference. This type of referencing is limited to 63 levels.

If a GO statement is used in a procedure with an entrance label to another procedure, this is not considered nesting but is a lateral transfer and does not change levels.

The externalized labels of the innermost procedure may be referenced outside the procedure sample, i.e., in the program proper. Any other labels are unknown outside this area of definition. Any of the labels (both unstarred and starred) may be referenced by a nested procedure. If, however, a reference is desired by the outer procedure to a label in the next immediately contained procedures, that label must be suffixed by an asterisk to reduce it to the level of the enclosing procedure.

Labels may be redefined on different levels. If more than two levels of nesting take place and a label defined in the innermost level is to be referenced in the outermost level, and then redefined successively upwards. Redefinition on higher levels of a nested procedure entry point is secured by adding as many asterisks to the entry as the number of procedure levels through which the label definition is to be carried.

Example of Nested Procedures:

```
Y*      PROC
B*      PROC
C*      PROC
  LA    A2, 1,, 1     . A2 is known to all
Z*      EQU  3        . Internal procedures
CC1     EQU  3        . Available only to C Procedure
CC2**   EQU  4        . Available to C,B,and Y Procedure
        END           . Terminates C Procedure
BB1     EQU  1        . Available in C and B Procedure
BB2*    EQU  2        . Available IN C, B, AND Y Procedure
Z*      EQU  Z        .
        END           . Terminates B Procedure
CC1     EQU  1        . Available in C,B,AND Y Procedure
CC4*    EQU  2        . Available in C,B,AND Y AND PROGRAM
 AA     EQU  12
        LX   Z, 4
        END           . Terminates Y The Outermost Procedure
```

The label Z is brought from the innermost to the outermost by a series of EQU directives. Nesting is time consuming and should be avoided when possible.

20.11.9. NAME

The NAME line has 3 functions. It provides a local reference point within a given procedure or function. It acts as an alternate entrance(s) into the procedure or function. In any case, it must be located between the PROC or FUNC line and its respective END line. The third function of the NAME line is that it may give a value to a procedure. This value is written as the operand of the NAME line and becomes meaningful as the 0th subfield, 0th field if and only if the procedure is entered at this name line. If such a value exists, this counts as an additional field to the procedure. Additional subfields may be added to this 0th field at the time of call as shown below: The paraform label (a,*b) will produce a 1 if that subfield in the call is preceded by an *. If not, it will produce a 0. Paraform labels may be used as operands on NAME lines only if NAME line is contained in a nested procedure.

```
SEE*  PROC  4          . MAX NO OF FIELDS IS 4
SAW*  NAME  2          . VALUE COUNTS AS A FIELD
      LA SEE(1,1),SEE(1,2),SEE(0,0),SEE(0,1)
      TLE SEE(2,1),SEE(3,1)
      GO  EYE                                    . (4)
EAR*  NAME  4
      BA  SEE(1,1),SEE(0,2),SEE(0,0),SEE(0,1)
      DO  SEE(3 ,*1) , +3                        . (5)
EYE*  NAME
      END
      .
      .  PROCEDURE CALLS
      SEE  16,CAT 17, DOG *43                    . (1)
      SAW  16,CAT 17, DOG  43                    . (2)
      EAR,6,7 17                                 . (3)
```

Line 1 represents a call into the procedure via the PROC line. The subfields represented in the operand filed are SEE(1,1), SEE(1,2),SEE(2,1),SEE(3,1) and SEE(4,1) which will be substituted in corresponding places in the procedure. At this time SEE(0,0) and SEE(0,1) will have values of 0 since the entrance was not made at a NAME line. The DO statement will generate a +3 data word.

The Line (2) entrance will provide values for SEE(0,0),SEE(0,1), SEE(1,1),SEE(1,2),SEE(2,1),SEE(2,2), and SEE(3,1). In addition the paraform SEE(0,1) is considered 5, the second subfield of the zeroth field. SEE(0,0) is 2. The DO statement will generate no line of coding.

Line(3) causes entrance at the NAME line labeled EAR. SEE(0,0) is 4. SEE(0,1) is 6 and SEE(0,2) is 7. SEE(3,*1) is 0.

The label of a NAME line must be made external (LABEL*) if it is to be used as an entrance point or if it is referenced prior to the NAME line in which it was defined.

20.11.10. GO

This directive transfers control of the assembler to the label in the operand field. This label must be a bona fide NAME line label or PROC label. GO is used within a procedure or function and may be a legitimate directive instruction used in conjunction with a DO statement. Line (4) in the example under 20.11.10, is effectual only if entrance is made at SEE or SAW lines. In this case, the procedure is terminated after the TLE is generated.

```
DONC        PROC
D1*         NAME   0
D2*         NAME   1
            DO   DONC(0,0), = 0  ,  GO OUT
            SZ    CAT
OUT*        NAME
            END
```

If entrance is made at D1, no code will be generated. If entrance is made at D2, then "S Z CAT" will be generated.

20.11.11. FUNC

SLEUTH II enables the user to obtain a value at assembly time contingent upon a set of parameters. The value obtained is substituted for the reference call within the program.

The function is different from the procedure in that a value is calculated when a function is referenced and no object lines of coding are ever generated.

The general rules of definition are similar to the PROC. A FUNC directive must start the definition area. This line must have a label which may be starred. If this line is an entry point it must be starred. The delineation of code is terminated with an END directive which must have an OPERAND. This OPERAND field will be an expression whose evaluation will result in the proper quantity being substituted into the reference point in the program.

NAME lines may be alternate entry points into the FUNC. The labels associated with these NAME lines must be starred in this event. NAME lines may also be used as local reference points within the FUNC. Forward references should be avoided.

A single list of n subfields is used. The reference point is of the form LABEL (a,b,...n) where LABEL is the FUNC line label and a,b,...n are input values. This reference point can be found imbedded within an expression or can be the entire expression itself.

20.11.11. FUNC (Continued)

Label (0) is meaningful as a paraform if entry to the function is made through a NAME line. This input value is the operand of the NAME line. If no values are given and the label alone is coded as a paraform, it represents the total number of subfields submitted to the FUNC.

A particular subfield within the FUNC list is referenced within a FUNC by writing the FUNC label followed by one expression enclosed in parentheses. This expression specifies the ordinal number of the subfield within the list.

PROCS or FUNCS may be nested within a FUNC provided the procedure is not a line generating one. It is usually nested so that the ability to redefine labels at different levels is available. All the rules of nesting as specified in NESTED PROCS apply to FUNC.

An example of the function is the case where a particular calculation is made throughout the coding. Providing that all the values are known at assembly time, a FUNC could be coded. Assume in the following example that a = 1, b = 2, c = 3, and d = 4.

```
AVGCOS*  FUNC
A(1)     EQU     AVGCOS(1)*AVGCOS(2)
B(1)     EQU     AVGCOS(3)*AVGCOS(4)
C(1)     EQU     A(1) + B(1)
D(1)     EQU     AVGCOS(1)+AVGCOS(3)
         END     C(1)/D(1)
.
.
.
         LA  12,AVGCOS(1,2,3,4), , 016
. THIS LINE CONTAINS THE REFERENCE WHICH
. WILL CAUSE GENERATION OF VALUE AT
. ASSEMBLY TIME
```

20.11.11. FUNC (Continued)

An example of a FUNC source code statement is written below. It includes examples of FUNC structure, a nested procedure, and function references. The value produced by reference to SQRT FUNC will be the square root of the largest square which is less than or equal to the parameter provided in the reference.

```
                                                          LINE
SQRT* FUNC                                                . 1
A(1)  EQU    0                                            . 2
B(1)  EQU    0                                            . 3
C*    PROC   0                                            . 4
A*(1) EQU    A(1)+2*B(1)+1                                . 5
B*(1) EQU    B(1)+1                                       . 6
      END                                                 . 7
D     NAME                                                . 8
      C                                                   . 9
      DO   SQRT(1)>A(1)  , GO  D                          .10
      END    B(1)-(SQRT(1)<A(1))                          .11
.
.
.
   +  SQRT(64)                                            .12
   +  2*SQRT(13)                                          .13
      END                                                 .14
```

The actions taken by the assembler are listed below:

Lines 1-11 stored by the assembler for later reference.

Line 12 is a reference to SQRT FUNC with one parameter (64). The reference produces a line with octal 000000000010.

Line 1 is the entrance to FUNC.

Line 2 equates a value of zero to A(1).

Line 3 equates a value of zero to B(1).

Line 9 is a reference to C PROC.

Line 4 is the entrance to C PROC. The first zero operand expression indicates that no list is to be submitted to C PROC when referenced. The second zero operand expression indicates that no object coding will be produced by C PROC.

Line 5 equates a value to the label A(n). The value produced is a result of the operand expression, and will be an ascending sequence of squares $(1,4,9...e_n)$.

20.11.11. FUNC (Continued)

Line 6 equates a value to the label B(n). The value produced is a result of the operand expression, and will be an ascending sequence of square roots ($1,2,3,\ldots e_n$).

Line 7 terminates this iteration of C PROC.

Line 10 compares the value of the SQRT parameter (64) to the nth value of A. If it is greater, the GO line will be executed once. Assembly continues at Line 8.

Line 8 is a NAME entry point.

Line 9 references C PROC for the second iteration.
If the SQRT parameter value is not greater than the value of A, assembly continues at Line 11.

Line 11 terminates SQRT FUNC. The operand expression provides the value of SQRT FUNC for this reference.

Line 13 is another reference to SQRT FUNC. The execution is identical. The object line produced by this reference would contain an octal value 000000000006.

Line 14 terminates the assembly or program.

20.11.12. WRD

The WRD directive is used to indicate the object computer word size in bits. When an output word is generated, it must not exceed the stated output word size, or a truncation error will be noted. This limitation does not have effect during the evaluation of expressions, since values are limited only to the 1107 word size, 36 bits. Only when a "line item" is generated will the defined output word size be considered.

The format is:

```
WRD     e
```

where e is any expression with a value equal to or less than 36.

This directive must precede lines of symbolic code which are to be affected. If coded within a PROC, the procedure must be explicitly referenced by name to get the effect. If a PROC is used, do not code the second operand.

Example:

```
DEFS* PROC
      WRD  30
      END
```

After WRD is encountered by the assembler, its effect continues until another is encountered. The WRD directive cannot be used for programs to be executed on the 1107 or 1108.

20.11.13. CHAR

The CHAR directive is used to alter translation of the 1107 character set to an alternate set of 6-bit equivalents. The translation takes place any time the assembler encounters one or more characters enclosed by apostrophes. The format is:

CHAR $c_1,e_1,c_2,e_2,\ldots.c_n,e_n$

where for each pair of expresions, c is the value of the 1107 character to be replaced by e. The value of both c and e expressions must be between octal 0 and octal 77.

EXAMPLE:

```
CHAR  6,024,7,025,010,026
```

where used, the Fieldata characters "A", "B", and "C" would be given the values 024,025,026 respectively. Alternately, if

```
I  DO  3,  CHAR I+5, I+023
```

were used, it would have the same effect.

This directive should precede any reference. The rules for WRD placement are the same for this directive.

20.11.14. LIST,UNLIST

These two directives enable the programer to control the listing of the assembler. The LIST directive negates the effect of an "N" option or a previously used UNLIST directive which suppressed the printing. The "N" option applies to EXEC 8 only.

20.11.15. EQUF

If repeated reference is made to the M, X and J fields of an instruction word, it may be desirable to have one symbol represent these fields.

The EQUF directive builds a nonliteral line item which includes the M, X, J, H, and I subfields in the standard instruction form. A field definition may be created and referred to by label wherever needed. The line item in EQUF is ORed with the referencing line to create an instruction word.

The format of the EQUF directive is:

```
LABEL    EQUF    M,X,J
```

EXAMPLES:

```
FIELD  EQUF  *0100, *8, 14
       LA    A 4, FIELD
```

giving

```
10   00   04   00   0   000000   from LA line
00   16   00   10   3   000100   from EQUF
10   16   04   10   3   000100   output word
```

```
FSTR     EQU     20
FREGS    EQUF    FSTR+12, X4
         SA      A5,FREGS
```

giving

```
01   00   05   00   0   000000   from SA line
00   00   00   04   0   000040   from EQUF
01   00   05   04   0   000040   output word
```

```
ADDRA    EQUF    TABL, *B4, S1
         LA      A1, ADDRA
```

assuming that the relative address of TABL is 10164 then:

```
10   00   01   00   0   000000   from LA line
00   15   00   04   2   010164   from EQUF
10   15   01   04   2   010164   output word
```

20.12. COMMUNICATION BETWEEN FORTRAN AND SLEUTH II

20.12.1. COMMUNICATION BETWEEN FORTRAN AND SLEUTH II THROUGH USE OF COMMON AREAS

It is possible to communicate information between FORTRAN and SLEUTH elements in a program through use of common areas of data storage. For instance, if a FORTRAN element contained the following statement:

```
COMMON/TAG/A(500),B,C(5,2)
```

and it is wished that a SLEUTH II element have access to A and/or B and/or C, then the SLEUTH II element should contain the following coding in the declarative section:

```
TAG     INFO     6Δnn
```

where TAG must be the same as in the FORTRAN Common statement and "nn" is an integer from 0-31 specifying a location counter. In the body of the SLEUTH II element, there should be:

```
$(nn).

A       RES  500
B       RES  1
C       RES   10
```

The RES declaratives for A, B, and C must be the first entries under location counter "nn". Also the size and sequence of the arrays A, B, and C must agree in both the FORTRAN and SLEUTH II elements. However, names A, B, and C need not agree.

It is possible to follow this procedure for any number of named Common areas using a different location counter for each INFO statement. If you desire to use Blank Common, e.g.

```
COMMON A(500),B,C(5,2)
```

then the tag on the INFO line in the SLEUTH II element must be N$BLNK.

An automatic table length tag is generated by any INFO statement that defines a core DTABLE (Group 6) or magnetic drum table (Group 7). This tag consists of L followed by the first 5 characters of the table tag. The table length may be incremented at load time by using this table length tag in a TAL card. The length tag may also be referenced in the SLEUTH II coding.

20.12.2. COMMUNICATION BETWEEN FORTRAN AND SLEUTH II THROUGH USE OF ARGUMENT LISTS

A. General

In order to use Argument lists as a means of inter-element communication, it is necessary for the Programer to be familiar with the FORTRAN conventions for generating Subroutine and Function references.

When the compiler encounters a Call statement or a Function reference, the following sequence is generated:

```
LMJ     B11, TAG
        .
        .
        .
  ARGUMENT LIST
        .
        .
        .
NOP
```

where "TAG" is the name of a Function or Subroutine, and entries in the Argument list can be any of three types:

VAR

where "VAR" is a variable name and the compiler generates the address of that variable.

(nn)

where "nn" is a constant (either numeric or HOLLERITH (H.A.))

The compiler generates the address of a word containing the constant or the address of the first word of a Hollerith character string.

J xxL

where "xx" is a statement number corresponding to the number following a "$" in the Argument list.

Note: When a subroutine is referenced, values are returned through the Arguments, but when a Function is referenced a one-word result is returned in Register AØ, or two-word results (in AØ and A1) will be returned in the case of double precision functions.

20.12.2. COMMUNICATION BETWEEN FORTRAN AND SLEUTH II THROUGH USE OF ARGUMENT LISTS (Continued)

B. Communication Between SLEUTH II Subroutine (or Function) Called by FORTRAN Element

1. ARGUMENTS are accessed by using Register B11, e. g.

```
L     AØ,Ø,B11
```

will load the address of the First Argument while,

```
L     AØ,*Ø,B11
```

will load the actual contents of the Argument and

```
J     1, B11
```

will transfer control to the Statement Number specified by the second Argument in the Call Statement.

2. To exit normally from a SLEUTH II Subroutine (or Function) with n Arguments, code

```
J     n+1,B11
```

3. FORTRAN assumes that only Registers AØ-A5, and R1-R3 will be used by Subroutines or Functions. If any others are used, they must be saved and restored before exiting.

4. When writing a SLEUTH II Subroutine (or Function), the entry point Tag (that is, the name by which the Subroutine/Function will be referenced) must be affixed with an "*".

C. Communication Between FORTRAN Subroutine or Function Called By SLEUTH II element

1. The programer must imitate the coding sequence generated by FORTRAN (See Section A).

For example, if the FORTRAN Subroutine statement looked like this

```
SUBROUTINE SUB(A,B,$)
```

then the calling sequence might look like

```
LMJ     B11, SUB
 +       A
        (Ø)
 J      TAG
NOP
```

Note that, on lines containing Constants, the "+" is not used since this would prevent the generation of a literal.

20.12.2. COMMUNICATION BETWEEN FORTRAN AND SLEUTH II THROUGH USE OF ARGUMENT LISTS (Continued)

C. Communication Between FORTRAN Subroutine or Function Called By SLEUTH II Element (Continued)

2. The FORTRAN Subroutine/Function will assume that Registers AØ-A5 and R1-R3 are available for use and need not be saved and restored.

3. A reference to a FORTRAN Function will return a one-word result in AØ.

D. Availability of FORTRAN Library Routines to SLEUTH II Programer

1. All routines resident on the FORTRAN Library (e. g., SIN, COS, FOLION, LXB, etc.) are available to the SLEUTH II programer. All that need be done is to imitate the FORTRAN calling sequence to access these routines (See Section C.).

For example, to read an item from tape:

```
LMJ     B11,ITMIN
        ('FNAME')
  +     ITEM
  +     NN
  J     EOF
NOP
```

where "FNAME" is the file name and EOF is the label of an End-Of-File routine,

or to convert a word at address ADD from Binary to XS3 with Ignore Fill:

```
LMJ     B11,LBX
  +       ADD
          (Ø)
NOP
  S     AØ,ADD
```

2. The routines in FORLIB also assume that Registers AØ-A5 and R1-R3 are available without having to be saved.

3. It must be noted that those Functions defined as Built-In (MOD, MINØ, etc. - See Chapter 6 for complete list of Built-In Functions) may not be referenced by the above method, since references to these Functions in FORTRAN generate in-line code rather than jumps to Library Routines.

20.13. APPENDIX C. INSTRUCTION REPERTOIRE

APPENDIX C. INSTRUCTION REPERTOIRE

SLEUTH I				SLEUTH II	
INSTRUCTION	MNEMONIC	f	j	MNEMONIC	INSTRUCTION
Store Positive	STP	01	0–15	SA	Store A
Store Negative	STN	02	0–15	SN SNA	Store Negative A
Store Magnitude	STM	03	0–15	SM SMA	Store Magnitude A
Store R	STR	04	0–15	SR	Store R
Store Zero	STZ	05	0–15	SZ	Store Zero
Store B	STB	06	0–15	SX	Store X
Load Positive	LDP	10	0–17	LA	Load A
Load Negative	LDN	11	0–17	LN LNA	Load Negative A
Load Magnitude	LDM	12	0–17	LM LMA	Load Magnitude A
Load Negative Magnitude	LNM	13	0–17	LNMA	Load Negative Magnitude A
Add	ADD	14	0–17	AA	Add to A
Subtract	SUB	15	0–17	ANA	Add Negative A
Add Magnitude	ADM	16	0–17	AM AMA	Add Magnitude to A
Subtract Magnitude	SBM	17	0–17	ANM ANMA	Add Negative Magnitude to A
Add and Load	ADL	20	0–17	AU	Add Upper
Subtract and Load	SBL	21	0–17	ANU	Add Negative Upper
Block Transfer	BTR	22	0–17	BT	Block Transfer
Load R	LDR	23	0–17	LR	Load R
Add to B	ADB	24	0–17	AX	Add to X
Subtract from B	SBB	25	0–17	ANX	Add Negative to X
Load B Modifier Only	LBM	26	0–17	LXM	Load X Modifier
Load B	LDB	27	0–17	LX	Load X
Multiply Integer	MPI	30	0–17	MI	Multiply Integer
Multiply Single	MPS	31	0–17	MSI	Multiply Single Integer
Multiply Fractional	MPF	32	0–17	MF	Multiply Fractional
Divide Integer	DVI	34	0–17	DI	Divide Integer
Divide Single and Load	DVL	35	0–17	DSF	Divide Single Fractional
Divide Fractional	DVF	36	0–17	DF	Divide Fractional

20.13. APPENDIX C. INSTRUCTION REPERTOIRE (Continued)

SLEUTH I				SLEUTH II	
INSTRUCTION	MNEMONIC	f	j	MNEMONIC	INSTRUCTION
Selective Set	SSE	40	0–17	OR	Logical OR
Selective Complement	SCP	41	0–17	XOR	Logical Exclusive OR
Selective Clear	SCL	42	0–17	AND	Logical AND
Selective Substitute	SSU	43	0–17	MLU	Masked Load Upper
Selective Even Parity Test	SEP	44	0–17	TEP	Test Even Parity
Selective Odd Parity Test	SOP	45	0–17	TOP	Test Odd Parity
Test Modifier	TMO	47	0–17	TLEM TNGM	Test Less or Equal to Modifier Test Not Greater than Modifier
Test Zero	TZR	50	0–17	TZ	Test for Zero
Test Not Zero	TNZ	51	0–17	TNZ	Test for Non Zero
Test Equal	TEQ	52	0–17	TE	Test for Equal
Test Not Equal	TNE	53	0–17	TNE	Test for Not Equal
Test Less Than or Equal	TLE	54	0–17	TLE TNG	Test for Less or Equal Test for Not Greater
Test Greater Than	TGR	55	0–17	TG	Test for Greater
Test Within Limits	TWL	56	0–17	TW	Test for Within Range
Test Outside Limits	TOL	57	0–17	TNW	Test for Not Within Range
Test Positive	TPO	60	0–17	TP	Test for Positive
Test Negative	TNG	61	0–17	TN	Test for Negative
Search Equal	SEQ	62	0–17	SE	Search for Equal
Search Not Equal	SNE	63	0–17	SNE	Search for Not Equal
Search Less Than or Equal	SLE	64	0–17	SLE SNG	Search for Less or Equal Search for Not Greater
Search Greater Than	SGR	65	0–17	SG	Search for Greater
Search Within Limits	SWL	66	0–17	SW	Search for Within Range
Search Outside Limits	SOL	67	0–17	SNW	Search for Not Within Range
Index Jump	IXJP	70	*	JGD	Jump on Greater and Decrement
Masked Search Equal	MSEQ	71	00	MSE	Masked Search for Equal
Masked Search Not Equal	MSNE	71	01	MSNE	Masked Search for Not Equal
Masked Search Less than or Equal	MSLE	71	02	MSLE MSNG	Masked Search for Less or Equal Masked Search for not Greater
Masked Search Greater Than	MSGR	71	03	MSG	Masked Search for Greater
Masked Search Within Limits	MSWL	71	04	MSW	Masked Search for Within Range
Masked Search Outside Limits	MSOL	71	05	MSNW	Masked Search for Not Within Range

* *The j and a designators together serve to specify any of the 128 words of Film Store.*

20.13. APPENDIX C. INSTRUCTION REPERTOIRE (Continued)

SLEUTH I				SLEUTH II	
INSTRUCTION	MENMONIC	f	j	MNEMONIC	INSTRUCTION
Wait for Interrupt	WAIT	72	00	W	Wait
Return Jump	RTJP	72	01	SLJ	Store Location and Jump
Positive Bit Control Jump	PBJP	72	02	JPS	Jump on Positive and Shift
Negative Bit Control Jump	NBJP	72	03	JNS	Jump on Negative and Shift
Add Halves	ADDH	72	04	AH	Add Halves
Subtract Halves	SUBH	72	05	ANH	Add Negative Halves
Add Thirds	ADDT	72	06	AT	Add Thirds
Subtract Thirds	SUBT	72	07	ANT	Add Negative Thirds
Execute Remote Instruction	EXRI	72	10	EX	Execute
Load Memory Lockout Register	LMLR	72	11	LL	Load Lockout Register
Set Trace and Jump	STJP	72	12		
Disable I/O Interrupts and Jump	DIJP	72	13	PAIJ	Prevent all Interrupts and Jump
Single Right Circular Shift	SCSH	73	00	SSC	Single Shift Circular
Double Right Circular Shift	DCSH	73	01	DSC	Double Shift Circular
Single Right Logical Shift	SLSH	73	02	SSL	Single Shift Logical
Double Right Logical Shift	DLSH	73	03	DSL	Double Shift Logical
Single Right Arithmetic Shift	SASH	73	04	SSA	Single Shift Algebraic
Double Right Arithmetic Shift	DASH	73	05	DSA	Double Shift Algebraic
Scale Factor Shift	SFSH	73	06	LSC	Load Shift and Count
Zero Jump	ZRJP	74	00	JZ	Jump on Zero
Non-Zero Jump	NZJP	74	01	JNZ	Jump on Non Zero
Positive Jump	POJP	74	02	JP	Jump on Positive
Negative Jump	NGJP	74	03	JN	Jump on Negative
Console Selective Jump Unconditional Jump	CSJP JUMP	74	04	JK J	Jump on Keys Jump
Selective Stop Jump	SSJP	74	05	HKJ HJ	Halt on Keys and Jump Halt and Jump
No Operation	NOOP	74	06	NOP	No Operation
Enable I/O Interrupts and Jump	EIJP	74	07	AAIJ	Allow All Interrupts and Jump
Even Jump	EVJP	74	10	JNB	Jump on No Low Bit
Odd Jump	ODJP	74	11	JB	Jump on Low Bit
Modifier Jump	MOJP	74	12	JMGI	Jump Modifier Greater and Increment
Load Modifier and Jump	LMJP	74	13	LMJ	Load Modifier and Jump
Overflow Jump	OVJP	74	14	JO	Jump on Overflow
No Overflow Jump	NOJP	74	15	JNO	Jump on No Overflow

20.13. APPENDIX C. INSTRUCTION REPERTOIRE (Continued)

SLEUTH I				SLEUTH II	
INSTRUCTION	MNEMONIC	f	j	MNEMONIC	INSTRUCTION
Carry Jump	CYJP	74	16	JC	Jump on Carry
No Carry Jump	NCJP	74	17	JNC	Jump on No Carry
Initiate Input Mode	IIPM	75	00	LIC	Load Input Channel
Initiate Monitored Input Mode	IMIM	75	01	LICM	Load Input Channel and Monitor
Input Mode Jump	IMJP	75	02	JIC	Jump on Input Channel Busy
Terminate Input Mode	TIPM	75	03	DIC	Disconnect Input Channel
Initiate Output Mode	IOPM	75	04	LOC	Load Output Channel
Initiate Monitored Output Mode	IMOM	75	05	LOCM	Load Output Channel and Monitor
Output Mode Jump	OMJP	75	06	JOC	Jump on Output Channel Busy
Terminate Output Mode	TOPM	75	07	DOC	Disconnect Output Channel
Initiate Function Mode	IFNM	75	10	LFC	Load Function in Channel
Initiate Monitored Function Mode	IMFM	75	11	LFCM	Load Function in Channel and Monitor
Function Mode Jump	FMJP	75	12	JFC	Jump on Function in Channel
Force External Transfer	FEXT	75	13	AFC	Allow Function in Channel
Enable All External Interrupts	EAEI	75	14	AACI	Allow All Channel Interrupts
Disable All External Interrupts	DAEI	75	15	PACI	Prevent All Channel Interrupts
Enable Single External Interrupt	ESEI	75	16	ACI	Allow Channel Interrupt
Disable Single External Interrupt	DSEI	75	17	PCI	Prevent Channel Interrupt
Floating Add	FLAD	76	00	FA	Floating Add
Floating Subtract	FLSB	76	01	FAN	Floating Add Negative
Floating Multiply	FLMP	76	02	FM	Floating Multiply
Floating Divide	FLDV	76	03	FD	Floating Divide
Floating Point Unpack	FLUP	76	04	LUF	Load and Unpack Floating
Floating Point Standardize Pack	FLNP	76	05	LCF	Load and Convert to Floating
Floating Characteristic Difference Magnitude	FLCM	76	06	MCDU	Magnitude of Characteristic Difference to Upper
Floating Characteristic Difference	FLCD	76	07	CDU	Characteristic Difference to Upper

21. GLOSSARY

GLOSSARY

ABSOLUTE ADDRESS: (1) An address that is permanently assigned by the machine designer to a storage location; (2) a pattern of characters that identifies a unique location without further modification.

ADDRESS: An identification as represented by a name, label, or number, for a register, location in storage or any other data source or destination.

ALPHANUMERIC: Pertaining to a character set that contains both letters and digits and usually other characters such as punctuation marks. Synonymous with alphameric.

ANALYSIS: The methodical investigation of a problem, and the separation of the problem into smaller related units for further detailed study.

ARGUMENT: An independent variable.

ARITHMETIC SHIFT: A shift that does not affect the sign position.

ARITHMETIC UNIT: The unit of a computing system that contains the circuits that perform arithmetic operations.

ARRAY: An arrangement of elements in one or more dimensions.

ASSEMBLE: To prepare a machine language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

AUTOMATIC DATA PROCESSING: The performance of operations upon information by a system of electronic or electrical machine so interconnected and interacting as to reduce to a minimum the need for human assistance or intervention.

BIAS: The amount by which the average of a set of values departs from a reference value.

BINARY: Pertaining to a characteristic or property involving a selection, choice or condition in which there are two possibilities.

BINARY-CODED DECIMAL NOTATION: A positional notation in which the individual decimal digits expressing a number in decimal notation are each represented by a binary numeral; BCD.

BINARY SEARCH: A dichotomizing search in which the number of items of the set is divided into two equal parts at each step of the process.

BISTABLE: Pertaining to a device capable of assuming either one of two stable states.

GLOSSARY (Continued)

BIT: An abbreviation of binary digit; a single character in a binary number.

BLOCK: A set of things, such as words, characters, or digits handled as a unit.

BUFFER: Intermediate storage area used when an item is being transferred from file to computer memory. Data conversion may be performed during the transfer from buffer to the memory.

CENTRAL PROCESSING UNIT: A unit of a computer that includes the circuits controlling the interpretation and execution of instructions.

CHARACTER: A letter, digit, or other symbol that is used as part of the organization, control, or representation of data.

CLEAR: To place one or more storage locations into a prescribed state, usually zero or the space character.

CODE: To represent data or a computer program in a symbolic form that can be accepted by a data processor.

COMPILE: To prepare a machine language program from a computer program written in another programing language by making use of the overall logic structure of the program, or generating more than one machine instruction for each symbolic statement, or both, as well as performing the function of an assembler.

COMPUTER, ANALOG: A computer which represents variables by physical analogies. Thus any computer which solves problems by translating physical conditions such as flow, temperature, pressure, angular position, or voltage into related mechanical or electrical quantities and uses mechanical or electrical equivalent circuits as an analog for the physical phenomenon being investigated.

COMPUTER, DIGITAL: A computer which processes data represented by combinations or discontinuous data as compared with an analog computer for continuous data.

CONSOLE: That part of a computer used for communication between the operator or maintenance engineer and the computer.

CONSTANT: (1) The quantities or messages, which will be present in the machine and available as data for the program and which, usually are not subject to change with time. (2) Information whose value or meaning is fixed for the program to which it belongs.

GLOSSARY (Continued)

CONTROL UNIT: In a digital computer, those parts that effect the retrieval of instructions in proper sequence, the interpretation of each instruction, and the application of the proper signals to the arithmetic unit and other parts in accordance with this interpretation.

CONVERT: To change numerical information from one number base to another, and/or from some form of fixed point to some form of floating point representation or vice versa.

DATA: A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation or processing by humans or automatic means.

DATA BANK: A comprehensive collection of libraries of data.

DATA PROCESSING: The execution of a systematic sequence of operations performed upon data.

DEBUG: To detect, locate and remove mistakes from a routine or malfunctions from a computer.

DECK: A collection of punched cards.

DECODE: Ascertain the intended meaning of the individual characters or groups of characters in the pseudo-coded program.

DOUBLE PRECISION: Pertaining to the use of two computer words to represent a number.

DUMMY: An artificial address, instruction, word, item, or other unit of information inserted solely to fulfill certain prescribed requirements, such as dummy parameters, without affecting operations.

DUMP: To copy the contents of all or part of a storage, usually from an internal storage into an external storage.

EDIT: To modify the form or format of data.

EFFECTIVE ADDRESS: The address that is derived by applying any specified indexing or indirect addressing rules to the specified address and that is actually used to identify the current operand.

ENTRY POINT: In a routine, any place to which control can be passed.

FIELD: In a record, a specified area used for a particular category of data.

FILE: An organized collection of information on external devices treated as a unit. For example, one line of an invoice may form an item, a complete invoice may form a record, and the complete set of such records may form a file, the collection of inventory control files may form a library and the libraries used by an organization are known as its data bank.

GLOSSARY (Continued)

FIXED POINT REPRESENTATION: A positional representation in which each number is represented by a single set of digits, the position of the radix point being fixed with respect to one end of the set, according to some convention.

FLOWCHART: A graphical representation for the definition, analysis, or solution of a problem, in which symbols are used to represent operations, data, flow, equipment, etc.

FORMAT: The arrangement of data.

GENERATOR, PROGRAM: A program which permits a computer to write other programs automatically.

HARDWARE: Physical equipment used for processing data; e.g. mechanical, magnetic, electrical, or electronic devices.

HOLLERITH: Pertaining to a particular type of code or punched card utilizing 12 rows per column and usually 8Ø columns per card.

I/O: Input/Output

INDEX: A symbol or a number used to identify a particular quantity in an array of similar quantities.

INITIALIZE: To set counters, switches, and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.

INPUT: The transfer of data from an external storage medium into the internal storage of the computer.

INSTRUCTION: A statement that specifies an operation and the values or locations of its operands. It, as a unit, causes the computer to perform the operation on the indicated quantities.

INTERRUPT: To stop a process in such a way that it can be resumed.

JOB: A specified group of tasks prescribed as a unit of work for a computer. A job usually includes all necessary computer programs, linkages, files, and instructions to the operating system.

JUMP: A departure from the normal sequence of executing instructions in a computer; transfer.

JUSTIFY: (1) To adjust the printing positions of characters on a page so the lines have the desired length and both the left and right hand margins are regular. (2) By extension, to shift the contents of a register so the most or the least significant digit is at some specified position in the register.

KEY: One or more characters within an item of data that are used to identify it or control its use.

KEYPUNCH: A keyboard actuated device that punches holes in a card to represent data.

LIBRARY: A collection of related files.

LOAD: In programing, to enter data into storage or working registers.

LOOP: A sequence of instructions that is executed repeatedly until a terminal condition prevails.

MAGNETIC CORE: A configuration of magnetic material that is, or is intended to be, placed in a special relationship to current-carrying conductors and whose magnetic properties are essential to its use. It may be made of such material as iron, iron oxide, or ferrite and in such shapes as wires, tapes, toroids or thin film.

MAGNETIC DISK: A flat circular plate with a magnetic surface on which data can be stored by selective magnetization of portions of the flat surface.

MAGNETIC DRUM: A right circular cylinder with a magnetic surface on which data can be stored by selective magnetization of portions of the curved surface.

MAGNETIC TAPE: A tape with a magnetic surface on which data can be stored by selective polarization of portions of the surface.

MAGNETIC THIN FILM: A layer of magnetic material, usually less than one micron thick, often used for logic or storage elements.

MASK: A pattern of characters that is used to control the retention or elimination of portions of another pattern of characters.

MATRIX: (1) In mathematics, a two-dimensional rectangular array of quantities. (2) In computers, a logic network in the form of an array of input leads and output leads with logic elements connected at some of their intersections.

MERGE: To combine items from two or more similarly ordered sets into one set that is arranged in the same order.

MICROSECOND: One-millionth of a second.

MILLISECOND: One-thousandth of a second.

MNEMONIC SYMBOL: A symbol chosen to assist the human memory, e.g. an abbreviation such as "mpy" for "multiply."

MONITOR: Software or hardware that observes, supervises, controls or verifies the operations of a system.

GLOSSARY (Continued)

MULTIPLE PUNCHING: Punching more than one hole in the same column on a punched card by means of more than one keystroke.

MULTIPROGRAMING: Pertaining to the concurrent execution of two or more programs by a computer.

NEST: To imbed subroutines or data in other subroutines or data at a different hierarchal level such that the different levels of routines or data can be executed or accessed recursively.

OBJECT PROGRAM: A fully compiled or assembled program that is ready to be loaded into the computer.

OFF-LINE: Pertaining to equipment or devices not under direct control of the central processing unit.

ON-LINE: Pertaining to equipment or devices under direct control of the central processing unit.

OPEN SHOP: Pertaining to the operation of a computer facility where computer programing, coding, and operating can be performed by any qualified employee of the organization.

OPERAND: That which is operated upon.

OPERATION: A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result for any permissible combination of operands.

PACK: To compress data in a storage medium by taking advantage of known characteristics of the data, in such a way that the original data can be recovered.

PARAMETER: A variable that is given a constant value for a specific purpose or process.

PERIPHERAL EQUIPMENT: Any unit of equipment distinct from the central processing unit, which may provide the system with outside communication.

PRECISION: The degree of discrimination with which a quantity is stated.

PROGRAM: A series of actions proposed in order to achieve a certain result.

RADIX POINT: The real or implied character that separates the digits associated with the integral part of a numeral from those associated with the fractional part.

READ: To acquire or interpret data from a storage device, a data medium, or any other source.

GLOSSARY (Continued)

RECORD: A collection of related items of data, treated as a unit.

ROLL IN: To restore in main storage data which had previously been transferred from main storage to auxiliary storage.

ROLL OUT: To record on an auxiliary storage device the contents of main storage.

RUN: A single, continuous performance of a computer routine.

SCALE: To examine sequentially part by part.

SEARCH KEY: Data to be compared to specific parts of each item for the purpose of conducting a search.

SENTINEL: A symbol marking the beginning or the end of some piece of information such as a field, item, block, tape, etc.

SERIAL: Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.

SERVICE ROUTINE: A routine in general support of the operation of a computer, e.g., an input/output, diagnostic, tracing, or monitoring routine.

SHIFT: A movement of data to the right or left.

SIGNIFICANT DIGIT: A digit that is needed for a certain purpose, particularly one that must be kept to preserve a specific accuracy or precision.

SOFTWARE: A set of computer programs, procedures, rules, and possible associated documentation concerned with the operation of a data processing system, e.g., compilers, library routines, manuals, circuit diagrams.

SORT: To segregate items into groups according to some definite rules.

SOURCE LANGUAGE: The language in which the programer codes and which is translated by the compiler into the object language which the computer can comprehend.

STORAGE ALLOCATION: The assignment of blocks of data to specified blocks of storage.

STRAIGHT LINE CODING: Coding in which loops are avoided by the repetition of parts of the coding when required.

SYMBOLIC ADDRESS: An address expressed in symbols convenient to the programer.

GLOSSARY (Continued)

SYMBOLIC CODING: Coding that uses machine instructions with symbolic addresses.

SYNTAX: The structure of expressions in a language.

SYSTEM, ELECTRONIC DATA PROCESSING: The general term used to define a system for data processing by means of machines utilizing electronic circuitry at electronic speed as opposed to electromechanical equipment.

TABLE LOOK-UP: A procedure for obtaining the function value corresponding to an argument from a table of function values.

WORD: A character string or a bit string considered as an entity.

WRITE: To record data in a storage device or a data medium.

BIBLIOGRAPHY

UNIVAC 1107 MANUALS:

Central Computer	UP-2463 Rev. 2
EXEC I	UP-2577 Rev. 2
FORTRAN IV Fundamentals	UP-7536 Rev. 1
FORTRAN IV Library Subroutines	UP-3947
FORTRAN IV Programmers Guide	UP-3540 D
FORTRAN IV Programmers Reference Manual	UP-3569 Rev. 1
LION	UP-2581 Rev. 2
PROCS General Discussion	UP-2593.33
SLEUTH I	UP-2574 Rev. 2
SLEUTH II	UP-3670 Rev. 1
SORT/MERGE I	UP-3958

UNIVAC 1108 MANUALS:

FORTRAN V Programmers Reference	UP-4060 Rev. 1
STAT-PACK Program Abstracts	UP-4041 Rev. 2
MATH-PACK Program Abstracts	UP-4051
Assembler Procedures	UP-4042
EXEC 8 Programmers Reference	UP-4144 Rev. 1
1106/1108 Assembler Programmers Reference	UP-4040 Rev. 2

BOOKS:

McCracken, Daniel D., A Guide To FORTRAN IV Programming, John Wiley & Sons, Inc., New York, New York, 1968.

MISCELLANEOUS:

Bureau of the Census Memoranda, Notes, Training Material, and Computer Systems Manual.

PENN STATE UNIVERSITY LIBRARIES
A0000708999443